

Distributed Signal Processing via Chebyshev Polynomial Approximation

David I Shuman, Pierre Vanderghenst, and Pascal Frossard

Ecole Polytechnique Fédérale de Lausanne (EPFL)

Signal Processing Laboratory

CH-1015 Lausanne, Switzerland

{david.shuman, pierre.vanderghenst, pascal.frossard}@epfl.ch

Abstract—Unions of graph multiplier operators are an important class of linear operators for processing signals defined on graphs. We present a novel method to efficiently distribute the application of these operators. The proposed method features approximations of the graph multipliers by shifted Chebyshev polynomials, whose recurrence relations make them readily amenable to distributed computation. We demonstrate how the proposed method can be applied to distributed processing tasks such as smoothing, denoising, inverse filtering, and semi-supervised classification, and show that the communication requirements of the method scale gracefully with the size of the network.

Index Terms—Chebyshev polynomial approximation, denoising, distributed optimization, learning, regularization, signal processing on graphs, spectral graph theory

I. INTRODUCTION

In distributed signal processing tasks, the data to be processed is physically separated and cannot be transmitted to a central processing entity. This separation may be due to engineering limitations such as the limited communication range of wireless sensor network nodes, privacy concerns, or engineering design considerations. Even when high-dimensional data can be processed centrally, for example, it may be more efficient to process it with parallel computing. It is therefore important to develop distributed data processing algorithms that balance the trade-offs between performance, communication bandwidth, and computational complexity (speed).

For concreteness, we focus throughout the paper on distributed processing examples in wireless sensor networks; however, the problems we consider could arise in a number of different settings. Due to the limited communication range of wireless sensor nodes, each sensor node in a large network is likely to communicate with only a small number of other nodes in the network. To model the communication patterns, we can write down a graph with each vertex corresponding to a sensor node and each edge corresponding to a pair of nodes that communicate. Moreover, because the communication graph is a function of the distances between nodes, it often captures

spatial correlations between sensors' observations as well. That is, if two sensors are close enough to communicate, their observations are likely to be correlated. We can further specify these spatial correlations by adding weights to the edges of the graph, with higher weights associated to edges connecting sensors with closely correlated observations. For example, it is common to construct the graph with a thresholded Gaussian kernel weighting function based on the physical distance between nodes, where the weight of edge e connecting nodes i and j that are a distance $d(i, j)$ apart is

$$w(e) = \begin{cases} \exp\left(-\frac{[d(i, j)]^2}{2\sigma^2}\right) & \text{if } d(i, j) \leq \kappa \\ 0 & \text{otherwise} \end{cases}, \quad (1)$$

for some parameters σ and κ .

We consider sensor networks whose nodes can only send messages to their local neighbors (i.e., they cannot communicate directly with a central entity). Much of the literature on distributed signal processing in such settings (see, e.g., [1]-[4] and references therein) focuses on coming to an agreement on simple features of the observed signal (e.g., consensus averaging, parameter estimation). We are more interested in processing the full function in a distributed manner, with each node having its own objective. Some example tasks under this umbrella include:

- *Distributed denoising* – In a sensor network of N sensors, a noisy N -dimensional signal is observed, with each component of the signal corresponding to the observation at one sensor location. Using the prior knowledge that the denoised signal should be smooth or piecewise smooth with respect to the underlying weighted graph structure, the sensors' task is to denoise each of their components of the signal by iteratively passing messages to their local neighbors and performing computations.
- *Distributed semi-supervised learning / transductive classification* – A class label is associated with each sensor node; however, only a small number of nodes in the network have knowledge of their labels. The cooperative task is for each node to learn its label by iteratively passing messages to its local neighbors and performing computations.

These and similar tasks have been considered in centralized settings in the relatively young field of signal processing on graphs. For example, [5]-[7] consider general regularization

Part of the work reported here was presented at the *IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*, June 2011, Barcelona, Spain.

This work was supported in part by FET-Open grant number 255931 UNLocX and FNS grant number 200021-118230.

The authors would also like to thank Jalal Fadili for his help deriving the bound in Proposition 5, and Javier Pérez-Trufero for his help producing some of the graphics in this paper.

frameworks on weighted graphs; [8]-[15] present graph-based semi-supervised learning methods; and [16]-[19] consider regularization and filtering on weighted graphs for image and mesh processing. Spectral regularization methods for ill-posed inverse problems (see, e.g., [20] and references therein) are also closely related.

Less work has been devoted to such tasks in distributed settings; reference [21] considers denoising via wavelet processing and [22] presents a denoising algorithm that projects the measured signal onto a low-dimensional subspace spanned by smooth functions. References [23]-[26] consider different distributed regression problems.

Our main contributions in this paper are i) to show that a key component of many distributed signal processing tasks is the application of linear operators that are unions of graph Fourier multipliers or generalized graph multiplier operators (to be defined in detail in Sections III and V, respectively); and ii) to present a novel method to efficiently distribute the application of the graph multiplier operators to high-dimensional signals.

To elaborate a bit, graph Fourier multiplier operators are the graph analog of filter banks, one of the most commonly used tools in digital signal processing. Multiplying a signal on the graph by one of these matrices is analogous to reshaping the signal's frequencies by multiplying it by a filter in the Fourier domain in classical signal processing. The crux of our novel distributed computational method is to approximate each graph Fourier multiplier by a truncated Chebyshev polynomial expansion. In a centralized setting, [27] shows that the truncated Chebyshev polynomial expansion efficiently approximates the application of a spectral graph wavelet transform, which is a specific example of a union of graph Fourier multipliers. In [28], we extend the Chebyshev polynomial approximation method to the general class of unions of graph Fourier multiplier operators, and show how the recurrence properties of the Chebyshev polynomials also enable distributed application of these operators. The communication requirements for distributed computation using this method scale gracefully with the number of sensors in the network (and, accordingly, the size of the signals). In this paper, an extended version of [28], we also generalize graph Fourier multiplier operators to generalized graph multiplier operators, provide theoretical bounds on the approximation error, illustrate the application of our framework in distributed smoothing, denoising, inverse filtering, and semi-supervised learning tasks, and compare our method to some alternative distributed computation methods.

The remainder of the paper is as follows. In the next section, we provide some background from spectral graph theory. In Section III, we introduce graph Fourier multiplier operators and show how they can be efficiently approximated with shifted Chebyshev polynomials in a centralized setting. We then discuss the distributed computation of quantities involving these operators in Section IV. We generalize the framework in Section V, and provide application examples in Section VI. In Section VII, we compare our proposed method with some alternative distributed computation methods. Section VIII concludes the paper.

II. SPECTRAL GRAPH THEORY

Before proceeding, we introduce some basic notations and definitions from spectral graph theory [29]. Throughout, we use bold font to denote matrices and vectors, and we denote the n^{th} component of a vector \mathbf{f} by either $f(n)$, $(\mathbf{f})_n$, or f_n . We model the communication network with an undirected, weighted graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, w\}$, which consists of a set of vertices \mathcal{V} , a set of edges \mathcal{E} , and a weight function $w : \mathcal{E} \rightarrow \mathbb{R}^+$ that assigns a non-negative weight to each edge. We assume the number of nodes in the network, $N = |\mathcal{V}|$, is finite, and the graph is connected. The adjacency (or weight) matrix \mathbf{W} for a weighted graph \mathcal{G} is the $N \times N$ matrix with entries $\mathbf{W}_{m,n}$, where

$$\mathbf{W}_{m,n} = \begin{cases} w(e), & \text{if } e \in \mathcal{E} \text{ connects vertices } m \text{ and } n \\ 0, & \text{otherwise} \end{cases}.$$

Therefore, the weighted graph \mathcal{G} can be equivalently represented as the triplet $\{\mathcal{V}, \mathcal{E}, \mathbf{W}\}$. The degree of each vertex is the sum of the weights of all the edges incident to it. We define the degree matrix \mathbf{D} to have diagonal elements equal to the degrees, and zeros elsewhere. The non-normalized graph Laplacian is then defined as $\mathcal{L} := \mathbf{D} - \mathbf{W}$.

A signal or function $f : \mathcal{V} \rightarrow \mathbb{R}^N$ defined on the vertices of the graph may be represented as a vector $\mathbf{f} \in \mathbb{R}^N$, where the n^{th} component of the vector \mathbf{f} represents the function value at the n^{th} vertex in \mathcal{V} . For any $\mathbf{f} \in \mathbb{R}^N$, \mathcal{L} satisfies

$$(\mathcal{L}\mathbf{f})(m) = \sum_{m \sim n} \mathbf{W}_{m,n} \cdot (f(m) - f(n)),$$

where $m \sim n$ indicates vertices m and n are connected.

As the graph Laplacian \mathcal{L} is a real symmetric matrix, it has a complete set of orthonormal eigenvectors. We denote these by $\{\chi_\ell\}_{\ell=0,1,\dots,N-1}$. These eigenvectors have associated real, non-negative eigenvalues $\{\lambda_\ell\}_{\ell=0,1,\dots,N-1}$ satisfying $\mathcal{L}\chi_\ell = \lambda_\ell\chi_\ell$ for $\ell = 0, 1, \dots, N-1$. Zero appears as an eigenvalue with multiplicity equal to the number of connected components of the graph [29]. Therefore, without loss of generality, we assume the eigenvalues of the Laplacian of the connected graph \mathcal{G} to be ordered as

$$0 = \lambda_0 < \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_{N-1} := \lambda_{\max}.$$

Just as the classical Fourier transform is the expansion of a function f in terms of the eigenfunctions of the Laplace operator

$$\hat{f}(\omega) = \langle e^{i\omega x}, f \rangle = \int_{\mathbb{R}} f(x) e^{-i\omega x} dx,$$

the *graph Fourier transform* $\hat{\mathbf{f}}$ of any function $\mathbf{f} \in \mathbb{R}^N$ on the vertices of G is the expansion of \mathbf{f} in terms of the eigenfunctions of the graph Laplacian. It is defined by

$$\hat{f}(\ell) := \langle \chi_\ell, \mathbf{f} \rangle = \sum_{n=1}^N \chi_\ell^*(n) f(n), \quad (2)$$

where we adopt the convention that the inner product be conjugate-linear in the first argument. The *inverse graph*

Fourier transform is given by

$$f(n) = \sum_{\ell=0}^{N-1} \hat{f}(\ell) \chi_{\ell}(n). \quad (3)$$

III. CHEBYSHEV POLYNOMIAL APPROXIMATION OF GRAPH FOURIER MULTIPLIERS

In this section, we introduce graph Fourier multiplier operators, unions of graph Fourier multiplier operators, and a computationally efficient approximation to unions of graph Fourier multiplier operators based on shifted Chebyshev polynomials. All methods discussed here are for a centralized setting, and we extend them to a distributed setting in Section IV.

A. Graph Fourier Multiplier Operators

For a function f defined on the real line, a *Fourier multiplier operator* or *filter* Ψ reshapes the function's frequencies through multiplication in the Fourier domain:

$$\widehat{\Psi f}(\omega) = g(\omega) \hat{f}(\omega), \text{ for every frequency } \omega.$$

Equivalently, denoting the Fourier and inverse Fourier transforms by \mathcal{F} and \mathcal{F}^{-1} , we have

$$\begin{aligned} \Psi f(x) &= \mathcal{F}^{-1} \left(g(\omega) \mathcal{F}(f)(\omega) \right) (x) \\ &= \frac{1}{2\pi} \int_{\mathbb{R}} g(\omega) \hat{f}(\omega) e^{i\omega x} d\omega. \end{aligned} \quad (4)$$

We can extend this straightforwardly to functions defined on the vertices of a graph by replacing the Fourier transform and its inverse in (4) with the graph Fourier transform and its inverse, defined in (2) and (3). Namely, a *graph Fourier multiplier operator* is a linear operator $\Psi : \mathbb{R}^N \rightarrow \mathbb{R}^N$ that can be written as

$$\begin{aligned} \Psi \mathbf{f}(n) &= \mathcal{F}^{-1} \left(g(\lambda_{\ell}) \mathcal{F}(f)(\ell) \right) (n) \\ &= \sum_{\ell=0}^{N-1} g(\lambda_{\ell}) \hat{f}(\ell) \chi_{\ell}(n). \end{aligned} \quad (5)$$

Equivalently, we can write $\Psi = \sum_{\ell=0}^{N-1} g(\lambda_{\ell}) \chi_{\ell} \chi_{\ell}^*$. We refer to $g(\cdot)$ as the *multiplier*. A high-level intuition behind (5) is as follows. The eigenvectors corresponding to the lowest eigenvalues of the graph Laplacian are the “smoothest” in the sense that $|\chi_{\ell}(m) - \chi_{\ell}(n)|$ is small for neighboring vertices m and n . At the extreme is χ_0 , which is a constant vector ($\chi_0(m) = \chi_0(n)$ for all m and n). The inverse graph Fourier transform (3) provides a representation of a signal \mathbf{f} as a superposition of the orthonormal set of eigenvectors of the graph Laplacian. The effect of the graph Fourier multiplier operator Ψ is to modify the contribution of each eigenvector. For example, applying a multiplier $g(\cdot)$ that is 1 for all λ_{ℓ} below some threshold, and 0 for all λ_{ℓ} above the threshold is equivalent to projecting the signal onto the eigenvectors of the graph Laplacian associated with the lowest eigenvalues. This is analogous to low-pass filtering in the continuous domain. Section VI contains further intuition about and examples of graph Fourier multiplier operators. For more properties of the graph Laplacian eigenvectors, see [30] and references therein.

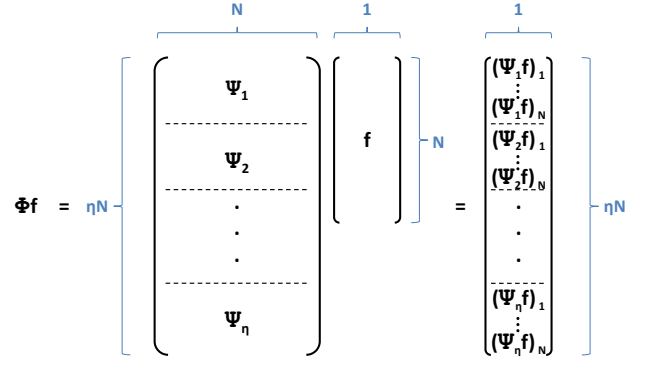


Fig. 1. Application of a union of graph Fourier multiplier operators.

B. Unions of Graph Fourier Multiplier Operators

In order for our distributed computation method of the next section to be applicable to a wider range of applications, we can generalize slightly from graph Fourier multipliers to *unions of graph Fourier multiplier operators*. A union of graph Fourier multiplier operators is a linear operator $\Phi : \mathbb{R}^N \rightarrow \mathbb{R}^N$ ($\eta \in \{1, 2, \dots\}$) whose application to a function $\mathbf{f} \in \mathbb{R}^N$ can be written as (see also Figure 1)

$$\begin{aligned} \Phi \mathbf{f} &= [\Psi_1; \Psi_2; \dots; \Psi_{\eta}] \mathbf{f} \\ &= [(\Psi_1 \mathbf{f})_1; \dots; (\Psi_1 \mathbf{f})_N; \dots; (\Psi_{\eta} \mathbf{f})_1; \dots; (\Psi_{\eta} \mathbf{f})_N] \\ &= [(\Phi \mathbf{f})_1; (\Phi \mathbf{f})_2; \dots; (\Phi \mathbf{f})_{\eta N}], \end{aligned}$$

where for every j , $\Psi_j : \mathbb{R}^N \rightarrow \mathbb{R}^N$ is a graph Fourier multiplier operator with multiplier $g_j(\cdot)$, and

$$\begin{aligned} (\Phi \mathbf{f})_{(j-1)N+n} &= \sum_{\ell=0}^{N-1} g_j(\lambda_{\ell}) \hat{f}(\ell) \chi_{\ell}(n), \\ &\text{for } j \in \{1, 2, \dots, \eta\}, n \in \{1, 2, \dots, N\}. \end{aligned} \quad (6)$$

C. The Chebyshev Polynomial Approximation

Exactly computing $\Phi \mathbf{f}$ requires explicit computation of the entire set of eigenvectors and eigenvalues of \mathcal{L} , which becomes computationally challenging as the size of the network, N , increases, even in a centralized setting. As discussed in detail in [27, Section 6], a computationally efficient approximation $\tilde{\Phi} \mathbf{f}$ of $\Phi \mathbf{f}$ can be computed by approximating each multiplier $g_j(\cdot)$ by a truncated series of shifted Chebyshev polynomials. Doing so circumvents the need to compute the full set of eigenvectors and eigenvalues of \mathcal{L} . We summarize this approach below.

For $y \in [-1, 1]$, the Chebyshev polynomials $\{T_k(y)\}_{k=0,1,2,\dots}$ are generated by

$$T_k(y) := \begin{cases} 1, & \text{if } k = 0 \\ y, & \text{if } k = 1 \\ 2yT_{k-1}(y) - T_{k-2}(y), & \text{if } k \geq 2 \end{cases}$$

These Chebyshev polynomials form an orthogonal basis for $L^2 \left([-1, 1], \frac{dy}{\sqrt{1-y^2}} \right)$. So every function h on $[-1, 1]$ that is square integrable with respect to the measure $dy/\sqrt{1-y^2}$

can be represented as $h(y) = \frac{1}{2}b_0 + \sum_{k=1}^{\infty} b_k T_k(y)$, where $\{b_k\}_{k=0,1,\dots}$ is a sequence of Chebyshev coefficients that depends on $h(\cdot)$. For a detailed overview of Chebyshev polynomials, including the above definitions and properties, see [31]–[33].

By shifting the domain of the Chebyshev polynomials to $[0, \lambda_{\max}]$ via the transformation $x = \frac{\lambda_{\max}}{2}(y + 1)$, we can represent each multiplier as

$$g_j(x) = \frac{1}{2}c_{j,0} + \sum_{k=1}^{\infty} c_{j,k} \bar{T}_k(x), \text{ for all } x \in [0, \lambda_{\max}], \quad (7)$$

where

$$\begin{aligned} \bar{T}_k(x) &:= T_k\left(\frac{x - \alpha}{\alpha}\right), \\ \alpha &:= \frac{\lambda_{\max}}{2}, \text{ and} \\ c_{j,k} &:= \frac{2}{\pi} \int_0^{\pi} \cos(k\theta) g_j\left(\alpha(\cos(\theta) + 1)\right) d\theta. \end{aligned} \quad (8)$$

For $k \geq 2$, the shifted Chebyshev polynomials satisfy

$$\bar{T}_k(x) = \frac{2}{\alpha}(x - \alpha)\bar{T}_{k-1}(x) - \bar{T}_{k-2}(x).$$

Thus, for any $\mathbf{f} \in \mathbb{R}^N$, we have

$$\bar{\mathbf{T}}_k(\mathcal{L})\mathbf{f} = \frac{2}{\alpha}(\mathcal{L} - \alpha\mathbf{I})(\bar{\mathbf{T}}_{k-1}(\mathcal{L})\mathbf{f}) - \bar{\mathbf{T}}_{k-2}(\mathcal{L})\mathbf{f}, \quad (9)$$

where $\bar{\mathbf{T}}_k(\mathcal{L}) \in \mathbb{R}^{N \times N}$ and the n^{th} element of $\bar{\mathbf{T}}_k(\mathcal{L})\mathbf{f}$ is given by

$$(\bar{\mathbf{T}}_k(\mathcal{L})\mathbf{f})_n := \sum_{\ell=0}^{N-1} \bar{T}_k(\lambda_{\ell}) \hat{f}(\ell) \chi_{\ell}(n). \quad (10)$$

Now, to approximate the operator Φ , we can approximate each multiplier $g_j(\cdot)$ by the first $K+1$ terms in its Chebyshev polynomial expansion (7). Then, for every $j \in \{1, 2, \dots, \eta\}$ and $n \in \{1, 2, \dots, N\}$, we have

$$\begin{aligned} &(\tilde{\Phi}\mathbf{f})_{(j-1)N+n} \\ &:= \left(\frac{1}{2}c_{j,0}\mathbf{f} + \sum_{k=1}^K c_{j,k} \bar{\mathbf{T}}_k(\mathcal{L})\mathbf{f} \right)_n \\ &\stackrel{(3),(10)}{=} \sum_{\ell=0}^{N-1} \left[\frac{1}{2}c_{j,0} + \sum_{k=1}^K c_{j,k} \bar{T}_k(\lambda_{\ell}) \right] \hat{f}(\ell) \chi_{\ell}(n) \\ &\approx \sum_{\ell=0}^{N-1} \left[\frac{1}{2}c_{j,0} + \sum_{k=1}^{\infty} c_{j,k} \bar{T}_k(\lambda_{\ell}) \right] \hat{f}(\ell) \chi_{\ell}(n) \\ &\stackrel{(7)}{=} \sum_{\ell=0}^{N-1} g_j(\lambda_{\ell}) \hat{f}(\ell) \chi_{\ell}(n) \\ &\stackrel{(6)}{=} (\Phi\mathbf{f})_{(j-1)N+n}. \end{aligned} \quad (11)$$

To recap, we propose to compute $\tilde{\Phi}\mathbf{f}$ by first computing the Chebyshev coefficients $\{c_{j,k}\}_{j=1,2,\dots,\eta; k=1,2,\dots,K}$ according to (8), and then computing the sum in (11). The computational benefit of the Chebyshev polynomial approximation arises in (11) from the fact the vector $\bar{\mathbf{T}}_k(\mathcal{L})\mathbf{f}$ can be computed recursively from $\bar{\mathbf{T}}_{k-1}(\mathcal{L})\mathbf{f}$ and $\bar{\mathbf{T}}_{k-2}(\mathcal{L})\mathbf{f}$ according to (9).

The computational cost of doing so is dominated by the matrix-vector multiplication of the graph Laplacian \mathcal{L} , which is proportional to the number of edges, $|\mathcal{E}|$ [27]. Therefore, if the underlying communication graph is sparse (i.e., $|\mathcal{E}|$ scales linearly with the network size N), it is far more computationally efficient to compute $\tilde{\Phi}\mathbf{f}$ than $\Phi\mathbf{f}$. Finally, we note that in practice, setting the Chebyshev approximation order K to around 20 results in $\tilde{\Phi}$ approximating Φ very closely in all of the applications we have examined.

IV. DISTRIBUTED CHEBYSHEV POLYNOMIAL APPROXIMATION

In the previous section, we showed that the Chebyshev polynomial approximation to a union of graph Fourier multipliers provides computational efficiency gains, even in a centralized computation setting. In this section, we discuss the second benefit of the Chebyshev polynomial approximation: it is easily distributable.

A. Distributed Computation of $\tilde{\Phi}\mathbf{f}$

We consider the following scenario. There is a network of N nodes, and each node n begins with the following knowledge:

- $f(n)$, the n^{th} component of the signal \mathbf{f}
- The identity of its neighbors, and the weights of the graph edges connecting itself to each of its neighbors
- The Chebyshev coefficients, $c_{j,k}$, for $j \in \{1, 2, \dots, \eta\}$ and $k \in \{0, 1, 2, \dots, K\}$. These can either be computed centrally according to (8) and then transmitted throughout the network, or each node can begin with knowledge of the multipliers, $\{g_j(\cdot)\}_{j=1,2,\dots,\eta}$, and precompute the Chebyshev coefficients according to (8)
- An upper bound on λ_{\max} , the largest eigenvalue of the graph Laplacian. This bound need not be tight, so we can precompute a bound such as $\lambda_{\max} \leq \max\{d(m) + d(n); m \sim n\}$, where $d(n)$ is the degree of node n [34]

The task is for each network node n to compute

$$\left\{ (\tilde{\Phi}\mathbf{f})_{(j-1)N+n} \right\}_{j=1,2,\dots,\eta} \quad (12)$$

by iteratively exchanging messages with its local neighbors in the network and performing some computations.

As a result of (11), for node n to compute the desired sequence in (12), it suffices to learn $\{(\bar{\mathbf{T}}_k(\mathcal{L})\mathbf{f})_n\}_{k=1,2,\dots,K}$. Note that $(\bar{\mathbf{T}}_1(\mathcal{L})\mathbf{f})_n = (\frac{1}{\alpha}(\mathcal{L} - \alpha\mathbf{I})\mathbf{f})_n$ and $\mathcal{L}_{n,m} = 0$ for all nodes m that are not neighbors of node n . Thus, to compute $(\bar{\mathbf{T}}_1(\mathcal{L})\mathbf{f})_n$, node n just needs to receive $f(m)$ from all neighbors m . So once all nodes send their component of the signal to their neighbors, they are able to compute their respective components of $\bar{\mathbf{T}}_1(\mathcal{L})\mathbf{f}$. In the next step, each node n sends the newly computed quantity $(\bar{\mathbf{T}}_1(\mathcal{L})\mathbf{f})_n$ to all of its neighbors, enabling the distributed computation of $\bar{\mathbf{T}}_2(\mathcal{L})\mathbf{f}$ according to (9). The iterative process of local communication and computation continues for K rounds until each node n has computed the required sequence $\{(\bar{\mathbf{T}}_k(\mathcal{L})\mathbf{f})_n\}_{k=1,2,\dots,K}$. In all, $2K|\mathcal{E}|$ messages of length 1 are required for every node n to compute its sequence of coefficients in (12) in a distributed

Algorithm 1 Distributed Computation of $\tilde{\Phi}\mathbf{f}$

Inputs at node n : $f_n, \mathcal{L}_{n,m} \forall m, \{c_{k,j}\}_{j=1,2,\dots,\eta; k=0,1,\dots,K}$, and λ_{\max}

Outputs at node n : $\left\{ \left(\tilde{\Phi}\mathbf{f} \right)_{(j-1)N+n} \right\}_{j=1,2,\dots,\eta}$

- 1: Set $(\bar{\mathbf{T}}_0(\mathcal{L})\mathbf{f})_n = f_n$
- 2: Transmit f_n to all neighbors $\mathcal{N}_n := \{m : \mathcal{L}_{n,m} < 0\}$
- 3: Receive f_m from all neighbors \mathcal{N}_n
- 4: Compute and store

$$(\bar{\mathbf{T}}_1(\mathcal{L})\mathbf{f})_n = \sum_{m \in \mathcal{N}_n \cup n} \frac{1}{\alpha} \mathcal{L}_{n,m} f_m - f_n$$

- 5: **for** $k = 2, \dots, K$ **do**
- 6: Transmit $(\bar{\mathbf{T}}_{k-1}(\mathcal{L})\mathbf{f})_n$ to all neighbors \mathcal{N}_n
- 7: Receive $(\bar{\mathbf{T}}_{k-1}(\mathcal{L})\mathbf{f})_m$ from all neighbors \mathcal{N}_n
- 8: Compute and store

$$(\bar{\mathbf{T}}_k(\mathcal{L})\mathbf{f})_n = \sum_{m \in \mathcal{N}_n \cup n} \frac{2}{\alpha} \mathcal{L}_{n,m} (\bar{\mathbf{T}}_{k-1}(\mathcal{L})\mathbf{f})_m - 2 (\bar{\mathbf{T}}_{k-1}(\mathcal{L})\mathbf{f})_n - (\bar{\mathbf{T}}_{k-2}(\mathcal{L})\mathbf{f})_n$$

- 9: **end for**
- 10: **for** $j \in \{1, 2, \dots, \eta\}$ **do**
- 11: Output

$$\left(\tilde{\Phi}\mathbf{f} \right)_{(j-1)N+n} = \frac{1}{2} c_{j,0} f_n + \sum_{k=1}^K c_{j,k} (\bar{\mathbf{T}}_k(\mathcal{L})\mathbf{f})_n$$

12: **end for**

fashion. This distributed computation process is summarized in Algorithm 1.

An important point to emphasize again is that although the operator Φ and its approximation $\tilde{\Phi}$ are defined through the eigenvectors of the graph Laplacian, the Chebyshev polynomial approximation helps the nodes apply the operator to the signal without explicitly computing (individually or collectively) the eigenvalues or eigenvectors of the Laplacian, other than the upper bound on its spectrum. Rather, they initially communicate their component of the signal to their neighbors, and then communicate simple weighted combinations of the messages received in the previous stage in subsequent iterations. In this way, information about each component of the signal \mathbf{f} diffuses through the network without direct communication between non-neighboring nodes.

B. Distributed Computation of $\tilde{\Phi}^*\mathbf{a}$

The application of the adjoint $\tilde{\Phi}^*$ of the Chebyshev polynomial approximate operator $\tilde{\Phi}$ can also be computed in a distributed manner. Let

$$\mathbf{a} = [\mathbf{a}_1; \mathbf{a}_2; \dots; \mathbf{a}_\eta] \in \mathbb{R}^{\eta N},$$

where $\mathbf{a}_j \in \mathbb{R}^N$. Then it is straightforward to show that

$$\left(\tilde{\Phi}^*\mathbf{a} \right)_n = \sum_{j=1}^{\eta} \left(\frac{1}{2} c_{j,0} \mathbf{a}_j + \sum_{k=1}^K c_{j,k} \bar{\mathbf{T}}_k(\mathcal{L})\mathbf{a}_j \right)_n. \quad (13)$$

Algorithm 2 Distributed Computation of $\tilde{\Phi}^*\mathbf{a}$

Inputs at node n : $\{a_j(n)\}_{j=1,2,\dots,\eta}, \mathcal{L}_{n,m} \forall m, \lambda_{\max}$, and $\{c_{k,j}\}_{j=1,2,\dots,\eta; k=0,1,\dots,K}$

Output at node n : $\left(\tilde{\Phi}^*\mathbf{a} \right)_n$

- 1: **for** $j = 1, 2, \dots, \eta$ **do**
- 2: Set $(\bar{\mathbf{T}}_0(\mathcal{L})\mathbf{a}_j)_n = a_j(n)$
- 3: **end for**
- 4: Transmit $\{a_j(n)\}_{j=1,2,\dots,\eta}$ to all neighbors $\mathcal{N}_n := \{m : \mathcal{L}_{n,m} < 0\}$
- 5: Receive $\{a_j(m)\}_{j=1,2,\dots,\eta}$ from all neighbors \mathcal{N}_n
- 6: **for** $j = 1, 2, \dots, \eta$ **do**
- 7: Compute and store

$$(\bar{\mathbf{T}}_1(\mathcal{L})\mathbf{a}_j)_n = \sum_{m \in \mathcal{N}_n \cup n} \frac{2}{\alpha} \mathcal{L}_{n,m} a_j(m) - 2a_j(n)$$

- 8: **end for**
- 9: **for** $k = 2, \dots, K$ **do**
- 10: Transmit $\{(\bar{\mathbf{T}}_{k-1}(\mathcal{L})\mathbf{a}_j)_n\}_{j=1,2,\dots,\eta}$ to all neighbors \mathcal{N}_n
- 11: Receive $\{(\bar{\mathbf{T}}_{k-1}(\mathcal{L})\mathbf{a}_j)_m\}_{j=1,2,\dots,\eta}$ from all neighbors \mathcal{N}_n
- 12: **for** $j = 1, 2, \dots, \eta$ **do**
- 13: Compute and store

$$(\bar{\mathbf{T}}_k(\mathcal{L})\mathbf{a}_j)_n = \sum_{m \in \mathcal{N}_n \cup n} \frac{2}{\alpha} \mathcal{L}_{n,m} (\bar{\mathbf{T}}_{k-1}(\mathcal{L})\mathbf{a}_j)_m - 2 (\bar{\mathbf{T}}_{k-1}(\mathcal{L})\mathbf{a}_j)_n - (\bar{\mathbf{T}}_{k-2}(\mathcal{L})\mathbf{a}_j)_n$$

- 14: **end for**
- 15: **end for**
- 16: Output

$$\left(\tilde{\Phi}^*\mathbf{a} \right)_n = \sum_{j=1}^{\eta} \left\{ \frac{1}{2} c_{j,0} a_j(n) + \sum_{k=1}^K c_{j,k} (\bar{\mathbf{T}}_k(\mathcal{L})\mathbf{a}_j)_n \right\}.$$

We assume each node n starts with knowledge of $a_j(n)$ for all $j \in \{1, 2, \dots, \eta\}$. For each $j \in \{1, 2, \dots, \eta\}$, the distributed computation of the corresponding term on the right-hand side of (13) is done in an analogous manner to the distributed computation of $\tilde{\Phi}\mathbf{f}$ discussed above. Since this has to be done for each j , $2K|\mathcal{E}|$ messages, each a vector of length η , are required for every node n to compute $\left(\tilde{\Phi}^*\mathbf{a} \right)_n$. The distributed computation of $\tilde{\Phi}^*\mathbf{a}$ is summarized in Algorithm 2.

C. Distributed Computation of $\tilde{\Phi}^*\tilde{\Phi}\mathbf{f}$

Using the property of the Chebyshev polynomials that

$$T_k(x)T_{k'}(x) = \frac{1}{2} [T_{k+k'}(x) + T_{|k-k'|}(x)],$$

we can write

$$\left(\tilde{\Phi}^*\tilde{\Phi}\mathbf{f} \right)_n = \left(\frac{1}{2} d_0 \mathbf{f} + \sum_{k=1}^{2K} d_k \bar{\mathbf{T}}_k(\mathcal{L})\mathbf{f} \right)_n$$

(see [27, Section 6.1] for a similar calculation and an explicit formula for the coefficients $\{d_k\}_{k=0,1,\dots,2K}$). Therefore, with each node n starting with $f(n)$ as in Section IV-A, the nodes can compute $\tilde{\Phi}^* \tilde{\Phi} \mathbf{f}$ in a distributed manner using $4K|\mathcal{E}|$ messages of length 1, with each node n finishing with knowledge of $(\tilde{\Phi}^* \tilde{\Phi} \mathbf{f})_n$.

D. Example: Distributed Smoothing

Perhaps the simplest example application of the distributed Chebyshev approximation method is distributed smoothing with the heat kernel as the graph Fourier multiplier. One way to smooth a signal $\mathbf{y} \in \mathbb{R}^N$ is to compute $\mathbf{H}_t \mathbf{y}$, where, for a fixed t , $(\mathbf{H}_t \mathbf{y})(n) := \sum_{\ell=0}^{N-1} e^{-t\lambda_\ell} \hat{y}(\ell) \chi_\ell(n)$. \mathbf{H}_t clearly satisfies our definition of a graph Fourier multiplier operator. In the context of a centralized image smoothing application, [18] discusses in detail the *heat kernel*, \mathbf{H}_t , and its relationship to classical Gaussian filtering. Similar to the example at the end of Section III-A, the main idea is that the multiplier $e^{-t\lambda_\ell}$ acts as a low-pass filter that attenuates the higher frequency (less smooth) components of \mathbf{y} .

Now, to perform distributed smoothing, we just need to compute $\tilde{\mathbf{H}}_t \mathbf{y}$ in a distributed manner according to Algorithm 1, where $\tilde{\mathbf{H}}_t$ is the shifted Chebyshev polynomial approximation to the graph Fourier multiplier operator \mathbf{H}_t . Each node starts with an observation y_n and finishes with $(\tilde{\mathbf{H}}_t \mathbf{y})_n$. We discuss more complicated application examples in Section VI.

V. GENERALIZED GRAPH MULTIPLIER OPERATORS

While defining the graph Fourier transform in terms of the eigenvectors of the graph Laplacian preserves a close analogy with the classical Fourier transform and leads to natural notions of smoothness for signals defined on graphs, the distributed computational methods of the previous section do not rely on specific properties of the graph Laplacian other than its real symmetric positive semi-definite nature. In this section, we generalize the definition of a graph Fourier multiplier operator by replacing \mathcal{L} with any real symmetric positive semi-definite matrix.

A. Definition and Equivalent Characterization of a Graph Multiplier Operator

Definition 1: Ψ is a graph multiplier operator with respect to the real symmetric positive semi-definite matrix \mathbf{P} if there exists a function $g : [0, \lambda_{\max}(\mathbf{P})] \rightarrow \mathbb{R}$ and a complete set $\{\chi_\ell\}_{\ell=0,1,\dots,N-1}$ of orthonormal eigenvectors of \mathbf{P} such that

$$\Psi = \sum_{\ell=0}^{N-1} g(\lambda_\ell) \chi_\ell \chi_\ell^*, \quad (14)$$

where $\{\lambda_\ell\}_{\ell=0,1,\dots,N-1}$ are the eigenvalues of \mathbf{P} .

We now provide equivalent characterizations of the class of graph multiplier operators with respect to a fixed matrix.

Proposition 1: The following are equivalent:

- (a) Ψ is a graph multiplier operator with respect to \mathbf{P} .
- (b) Ψ and \mathbf{P} are simultaneously diagonalizable by a unitary matrix; i.e., there exists a unitary matrix \mathbf{U} such that $\mathbf{U}^* \Psi \mathbf{U}$ and $\mathbf{U}^* \mathbf{P} \mathbf{U}$ are both diagonal matrices.

- (c) Ψ and \mathbf{P} commute; i.e., $\Psi \mathbf{P} = \mathbf{P} \Psi$.

Proof of Proposition 1: (a) implies (b) if we set the i^{th} column of \mathbf{U} to χ_{i-1} , and (b) implies (a) if we set χ_ℓ to the $(\ell + 1)^{\text{st}}$ column of \mathbf{U} and $g(\lambda_\ell)$ to the $(\ell + 1)^{\text{st}}$ diagonal element of $\mathbf{U}^* \Psi \mathbf{U}$. The equivalence between (b) and (c) is shown in [35, Corollary 4.5.18]. ■

B. Bound on the Approximation Error

Another interesting question is how closely a graph Fourier multiplier (or union of such operators) is approximated by its Chebyshev polynomial approximation. The following result, which bounds the spectral norm of their difference, is used in Section VI-B.

Proposition 2: Let Φ be a union of η generalized graph multiplier operators; i.e., it has the form given in (6), where $\{\chi_\ell\}_{\ell=0,1,\dots,N-1}$ are the eigenvectors of a real positive semi-definite matrix \mathbf{P} . Let $\tilde{\Phi}$ be the order K Chebyshev polynomial approximation of Φ . Define

$$B(K) := \max_{j=1,2,\dots,\eta} \left\{ \sup_{\lambda \in [0, \lambda_{\max}]} \{|g_j(\lambda) - p_j^K(\lambda)|\} \right\},$$

where λ_{\max} is the largest eigenvalue of \mathbf{P} , and $p_j^K(\cdot)$ is the order K Chebyshev polynomial approximation of $g_j(\cdot)$. Then

$$\|\Phi - \tilde{\Phi}\|_2 := \max_{\mathbf{f} \neq 0} \frac{\|(\Phi - \tilde{\Phi})\mathbf{f}\|_2}{\|\mathbf{f}\|_2} \leq B(K) \sqrt{\eta N}. \quad (15)$$

The proof of Proposition 2 is included in the Appendix.

Finally, note that when the multipliers $g_j(\cdot)$ are smooth, the Chebyshev approximations $p_j^K(\cdot)$ converge to the multipliers rapidly as K increases. In particular, the following proposition applies.

Proposition 3 (Theorem 5.14, [31]): If $g_j(\cdot)$ has $M + 1$ continuous derivatives for all j , then $B(K) = \mathcal{O}(K^{-M})$.

The Chebyshev polynomial approximation and distributed computation methods presented in Section III and IV also apply to these generalized graph multiplier operators. This generalization significantly extends the applicability of our distributed computation method, and is especially useful in distributed applications where we can first choose the eigenbasis to use in the transform, and then design the communication graph accordingly.

VI. ILLUSTRATIVE APPLICATIONS

In this section, we provide more detailed explanations of how our novel distributed approximation framework can be used in the context of a diverse set of distributed signal processing tasks in sensor networks. To be clear, our contribution here is not to suggest new signal processing techniques, but simply to show how a subset of the existing centralized techniques can be efficiently implemented in a distributed manner.

A. Denoising with Distributed Tikhonov Regularization

In this section, we consider the distributed denoising task discussed in Section I. To recall, we start with a noisy signal $\mathbf{y} \in \mathbb{R}^N$ that is defined on a graph of N sensors and has been corrupted by uncorrelated additive Gaussian noise. Through an iterative process of local communication and computation, we would like each sensor to end up with a denoised estimate of its component, f_n^0 , of the true underlying signal, \mathbf{f}^0 .

To solve this problem, we enforce *a priori* information that the target signal is smooth with respect to the underlying graph topology. To enforce the global smoothness prior, we consider the class of regularization terms $\mathbf{f}^T \mathcal{L}^r \mathbf{f}$ for $r \geq 1$. The resulting distributed regularization problem has the form

$$\underset{\mathbf{f}}{\operatorname{argmin}} \frac{\tau}{2} \|\mathbf{f} - \mathbf{y}\|_2^2 + \mathbf{f}^T \mathcal{L}^r \mathbf{f}. \quad (16)$$

To see intuitively why incorporating such a regularization term into the objective function encourages smooth signals (with $r = 1$ as an example), note that $\mathbf{f}^T \mathcal{L} \mathbf{f} = 0$ if and only if \mathbf{f} is constant across all vertices, and, more generally,

$$\mathbf{f}^T \mathcal{L} \mathbf{f} = \frac{1}{2} \sum_{n \in \mathcal{V}} \sum_{m \sim n} \mathbf{W}_{m,n} (f(m) - f(n))^2,$$

so $\mathbf{f}^T \mathcal{L} \mathbf{f}$ is small when the signal \mathbf{f} has similar values at neighboring vertices with large weights (i.e., it is smooth).

We now show how our novel method is useful in solving this distributed regularization problem.

Proposition 4: The solution to (16) is given by $\mathbf{R}\mathbf{y}$, where \mathbf{R} is a graph Fourier multiplier operator of the form (5), with multiplier $g(\lambda_\ell) = \frac{\tau}{\tau + 2\lambda_\ell}$.¹

The proof of Proposition 4 is included in the Appendix.

So, one way to do distributed denoising is to compute $\tilde{\mathbf{R}}\mathbf{y}$, the Chebyshev polynomial approximation of $\mathbf{R}\mathbf{y}$, in a distributed manner via Algorithm 1. We show this now with a numerical example. We place 500 sensors randomly in the $[0, 1] \times [0, 1]$ square. We then construct a weighted graph according to the thresholded Gaussian kernel weighting (1) with $\sigma = 0.074$ and $\kappa = 0.600$, so that two sensor nodes are connected if their physical separation is less than 0.075. We create a smooth 500-dimensional signal with the n^{th} component given by $f_n^0 = n_x^2 + n_y^2 - 1$, where n_x and n_y are node n 's x and y coordinates in $[0, 1] \times [0, 1]$. One instance of such a network and signal \mathbf{f}^0 are shown in Figure 2, and the eigenvectors of the graph Laplacian are shown in Figure 3.

Next, we corrupt each component of the signal \mathbf{f}^0 with uncorrelated additive Gaussian noise with mean zero and standard deviation 0.5. Then we apply the graph Fourier multiplier operator $\tilde{\mathbf{R}}$, the Chebyshev polynomial approximation to \mathbf{R} from Proposition 4, with $\tau = r = 1$ and $K = 15$. The multiplier and its Chebyshev polynomial approximations are shown in Figure 4, and the denoised signal $\tilde{\mathbf{R}}\mathbf{y}$ is shown in Figure 5. We repeated this entire experiment 1000 times, with a new random graph and random noise each time, and the average mean square error for the denoised signals was 0.013, as compared with 0.250 average mean square error for the noisy signals.

¹This filter $g(\lambda_\ell)$ is the graph analog of a first-order Bessel filter from classical signal processing of functions on the real line.

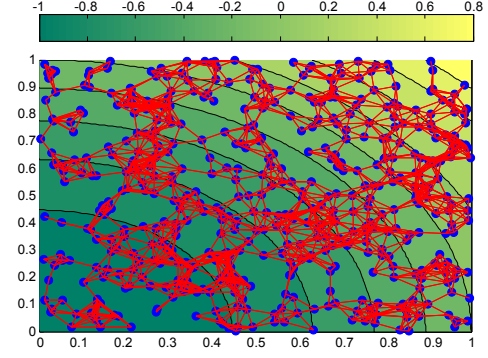


Fig. 2. A network of 500 sensors placed randomly in the $[0, 1] \times [0, 1]$ square. The background colors represent the values of the smooth signal \mathbf{f}^0 .

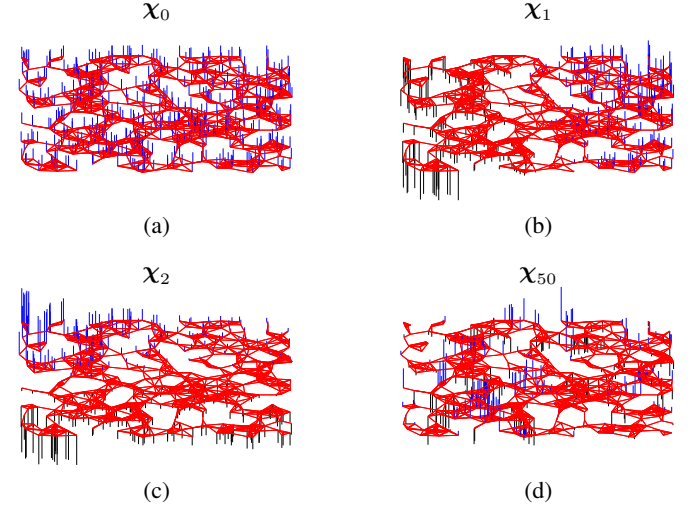


Fig. 3. Some eigenvectors of the Laplacian of the graph shown in Figure 2. The blue bars represent positive values and the black bars negative values. (a) χ_0 , the constant eigenvector associated with $\lambda_0 = 0$. (b) χ_1 , the Fiedler vector associated with the lowest strictly positive eigenvalue, nicely separates the graph into two components. (c) χ_2 is also a smooth eigenvector. (d) χ_{50} is far less smooth with some large differences across neighboring nodes.

B. Denoising with Distributed lasso

We now consider an alternative method of distributed denoising that is better suited to situations where we start with a prior belief that the signal is not globally smooth, but rather piecewise smooth, which corresponds to the signal being sparse in the spectral graph wavelet domain [27].

The spectral graph wavelet transform, Ξ , defined in [27], is precisely of the form of Φ in (6). Namely, it is composed of one multiplier, $h(\cdot)$, that acts as a low-pass filter to stably represent the signal's low frequency content, and J wavelet operators, defined by $g_j(\lambda_\ell) = g(t_j \lambda_\ell)$, where $\{t_j\}_{j=1,2,\dots,J}$ is a set of scales and $g(\cdot)$ is the wavelet multiplier that acts as a band-pass filter.

The most common way to incorporate a sparse prior in a centralized setting is to regularize via a weighted version of the *least absolute shrinkage and selection operator (lasso)* [36], also called *basis pursuit denoising* [37]:

$$\underset{\mathbf{a}}{\operatorname{argmin}} \frac{1}{2} \|\mathbf{y} - \Xi^* \mathbf{a}\|_2^2 + \|\mathbf{a}\|_{1,\mu}, \quad (17)$$

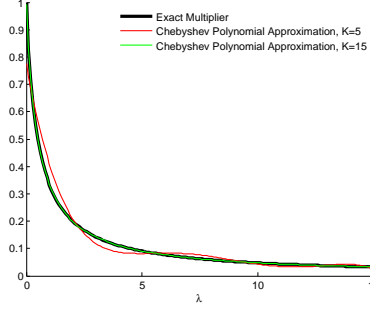


Fig. 4. The regularizing multiplier $\frac{\tau}{\tau + 2\lambda\ell}$ associated with the graph Fourier multiplier operator \mathbf{R} from Proposition 4. Here, $r = \tau = 1$. Shifted Chebyshev polynomial approximations to the multiplier are shown for different values of the approximation order K .

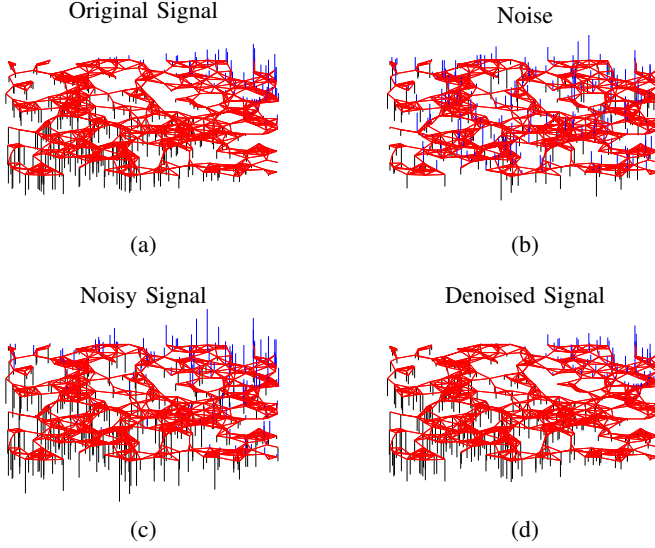


Fig. 5. A denoising example on the graph shown in Figure 2, using the regularizing multiplier shown in Figure 4. (a) The original signal $n_x^2 + n_y^2 - 1$, where n_x and n_y are the x and y coordinates of sensor node n . (b) The additive Gaussian noise. (c) The noisy signal \mathbf{y} . (d) The denoised signal $\tilde{\mathbf{R}}\mathbf{y}$.

where $\|\mathbf{a}\|_{1,\mu} := \sum_{i=1}^{N(J+1)} \mu_i |a_i|$ and $\mu_i > 0$ for all i . The optimization problem in (17) can be solved for example by iterative soft thresholding [38]. The initial estimate of the wavelet coefficients $\mathbf{a}^{(0)}$ is arbitrary, and at each iteration of the soft thresholding algorithm, the update of the estimated wavelet coefficients is given by

$$a_i^{(\beta)} = \mathcal{S}_{\mu_i \gamma} \left(\left(\mathbf{a}^{(\beta-1)} + \gamma \Xi \left[\mathbf{y} - \Xi^* \mathbf{a}^{(\beta-1)} \right] \right)_i \right),$$

$$i = 1, 2, \dots, N(J+1); \beta = 1, 2, \dots \quad (18)$$

where γ is the step size and $\mathcal{S}_{\mu_i \gamma}$ is the shrinkage or soft thresholding operator

$$\mathcal{S}_{\mu_i \gamma}(z) := \begin{cases} 0 & \text{if } |z| \leq \mu_i \gamma \\ z - \text{sgn}(z) \mu_i \gamma & \text{o.w.} \end{cases}.$$

The iterative soft thresholding algorithm converges to \mathbf{a}_* , the minimizer of (17), if $\gamma < \frac{2}{\|\Xi^*\|^2}$ [39]. The final denoised estimate of the signal is then given by $\Xi^* \mathbf{a}_*$.

We now turn to the issue of how to implement the above algorithm in a distributed fashion by sending messages between neighbors in the network. One option would be to

Algorithm 3 Distributed lasso

Inputs at node n : $y_n, \mathcal{L}_{n,m} \forall m, \{c_{k,j}\}_{j=1,2,\dots,J+1; k=0,1,\dots,K}, \lambda_{\max}, \gamma$, and $\{\mu_{(j-1)N+n}\}_{j=1,2,\dots,J+1}$
 Outputs at node n : y_{n*} , the denoised estimate of f_n^0

- 1: Arbitrarily initialize $\{(\tilde{\mathbf{a}}^{(0)})_{(j-1)N+n}\}_{j=1,2,\dots,J+1}$
- 2: Set $\beta = 1$
- 3: Compute and store $\{(\tilde{\Xi} \mathbf{y})_{(j-1)N+n}\}_{j=1,2,\dots,J+1}$ via Algorithm 1
- 4: **while** stopping criterion not satisfied **do**
- 5: Compute and store $\{(\tilde{\Xi} \tilde{\Xi}^* \tilde{\mathbf{a}}^{(\beta-1)})_{(j-1)N+n}\}_{j=1,2,\dots,J+1}$ via Algorithm 2, followed by Algorithm 1
- 6: **for** $j = 1, 2, \dots, J+1$ **do**
- 7: Compute and store $(\tilde{\mathbf{a}}^{(\beta)})_{(j-1)N+n}$

$$= \mathcal{S}_{(\mu_{(j-1)N+n})\gamma} \begin{pmatrix} \tilde{\mathbf{a}}_{(j-1)N+n}^{(\beta-1)} + \gamma (\tilde{\Xi} \mathbf{y})_{(j-1)N+n} \\ -\gamma (\tilde{\Xi} \tilde{\Xi}^* \tilde{\mathbf{a}}^{(\beta-1)})_{(j-1)N+n} \end{pmatrix}$$
- 8: **end for**
- 9: Set $\beta = \beta + 1$
- 10: **end while**
- 11: **for** $j = 1, 2, \dots, J+1$ **do**
- 12: Set $(\tilde{\mathbf{a}}_*)_{(j-1)N+n} = (\tilde{\mathbf{a}}^{(\beta)})_{(j-1)N+n}$
- 13: **end for**
- 14: Compute and store $y_{n*} = (\tilde{\mathbf{W}}^* \tilde{\mathbf{a}}_*)_n$ via Algorithm 2
- 15: Output y_{n*}

use the distributed lasso algorithm of [25], [26], which is a special case of the alternating direction method of multipliers [40, p. 253]. In every iteration of that algorithm, each node transmits its current estimate of *all* the wavelet coefficients to its local neighbors. With the spectral graph wavelet transform, that method requires $2|\mathcal{E}|$ total messages at every iteration, with each message being a vector of length $N(J+1)$. A method where the amount of communicated information does not grow with N (beyond the number of edges, $|\mathcal{E}|$) would be highly preferable.

The Chebyshev polynomial approximation of the spectral graph wavelet transform allows us to accomplish this goal. Our approach, which is summarized in Algorithm 3, is to approximate Ξ by $\tilde{\Xi}$, and use the distributed implementation of the approximate wavelet transform and its adjoint to perform iterative soft thresholding in order to solve

$$\underset{\tilde{\mathbf{a}}}{\operatorname{argmin}} \frac{1}{2} \|\mathbf{y} - \tilde{\Xi}^* \tilde{\mathbf{a}}\|_2^2 + \|\tilde{\mathbf{a}}\|_{1,\mu}. \quad (19)$$

In the first soft thresholding iteration, each node n must learn $(\tilde{\Xi} \mathbf{y})_{(j-1)N+n}$ at all scales j , via Algorithm 1. These coefficients are then stored for future iterations. In the β^{th} iteration, each node n must learn the $J+1$ coefficients of $\tilde{\Xi} \tilde{\Xi}^* \tilde{\mathbf{a}}^{(\beta-1)}$ centered at n , by sequentially applying the operators $\tilde{\Xi}^*$ and $\tilde{\Xi}$ in a distributed manner via Algorithms

2 and 1, respectively. When a stopping criterion for the soft thresholding is satisfied, the adjoint operator $\tilde{\Xi}^*$ is applied again in a distributed manner to the resulting coefficients $\tilde{\mathbf{a}}_*$, and node n 's denoised estimate of its signal is $(\tilde{\Xi}^* \tilde{\mathbf{a}}_*)_n$. The stopping criterion may simply be a fixed number of iterations, or it may be when $\left| (\tilde{\Xi}^* \tilde{\mathbf{a}}^{(\beta)})_n - (\tilde{\Xi}^* \tilde{\mathbf{a}}^{(\beta-1)})_n \right| < \epsilon$ for all n and some small ϵ . Finally, note that we could also optimize the weights $\boldsymbol{\mu}$ by performing distributed cross-validation, as discussed in [25], [26].

We now examine the communication requirements of this approach. Recall from Section IV that $2K|\mathcal{E}|$ messages of length 1 are required to compute $\tilde{\Xi} \mathbf{y}$ in a distributed fashion. Distributed computation of $\tilde{\Xi} \tilde{\Xi}^* \tilde{\mathbf{a}}^{(\beta-1)}$, the other term needed in the iterative thresholding update (18), requires $2K|\mathcal{E}|$ messages of length $J+1$ and $2K|\mathcal{E}|$ messages of length 1. The final application of the adjoint operator $\tilde{\Xi}^*$ to recover the denoised signal estimates requires another $2K|\mathcal{E}|$ messages, each a vector of length $J+1$. Therefore, the Chebyshev polynomial approximation to the spectral graph wavelet transform enables us to iteratively solve the weighted lasso in a distributed manner where the communication workload only scales with the size of the network through $|\mathcal{E}|$, and is otherwise independent of the network dimension N .

The reconstructed signal in Algorithm 3 is $\tilde{\Xi}^* \tilde{\mathbf{a}}_*$, where $\tilde{\mathbf{a}}_*$ is the solution to the lasso problem (19). A natural question is how good of an approximation $\tilde{\Xi}^* \tilde{\mathbf{a}}_*$ is to $\Xi^* \mathbf{a}_*$, where \mathbf{a}_* is the solution to the original lasso problem (17). The following proposition bounds the squared distance between these two quantities by a term proportional to the spectral norm of the difference between the exact and approximate spectral graph wavelet operators.

Proposition 5: $\|\tilde{\Xi}^* \tilde{\mathbf{a}}_* - \Xi^* \mathbf{a}_*\|_2^2 \leq C \|\tilde{\Xi} - \Xi\|_2$, where $\|\cdot\|_2$ is the spectral norm, and the constant $C = \frac{\|\mathbf{y}\|_2^3}{\min_i \mu_i}$.

Combining Proposition 5, whose proof is included in the Appendix, with (15), we have

$$\|\tilde{\Xi}^* \tilde{\mathbf{a}}_* - \Xi^* \mathbf{a}_*\|_2^2 \leq \frac{\|\mathbf{y}\|_2^3}{\min_i \mu_i} B(K) \sqrt{N(J+1)}. \quad (20)$$

Thus, as we increase the approximation order K , $B(K)$ and the right-hand side of (20) tend toward zero (at a speed dependent on the smoothness of the graph wavelet multipliers $g(\cdot)$ and $h(\cdot)$).

Finally, to illustrate the distributed lasso, we again consider a numerical example where we place 500 sensors randomly in the $[0, 1] \times [0, 1]$ square, with the same graph construction as the numerical example in Section VI-A. This time, however, the underlying signal is only piecewise-smooth, with the n^{th} component given by

$$f_n^0 = \begin{cases} -2n_x + 0.5, & \text{if } n_y \geq 1 - n_x \\ n_x^2 + n_y^2 + 0.5, & \text{if } n_y < 1 - n_x \end{cases}.$$

We again corrupt each component of the signal \mathbf{f}^0 with uncorrelated additive Gaussian noise with mean zero and standard deviation 0.5. We then solve problem (19) in a distributed manner using Algorithm 3. We use a spectral graph wavelet transform with 6 wavelet scales, with the kernels automatically designed by the spectral graph wavelets toolbox [41]. In

Algorithm 3, we run 300 soft thresholding iterations and take $\gamma = 0.2$, $\mu_i = 0.75$ for all the wavelet coefficients, and $\mu_i = 0.01$ for all the scaling coefficients.² We do not perform any distributed cross-validation to optimize the weights $\boldsymbol{\mu}$. We repeated this entire experiment 1000 times, with a new random graph and random noise each time.³ The average mean square errors were 0.250 for the noisy signals, 0.098 for the estimates produced by the Tikhonov regularization method (16), 0.088 for the denoised estimates produced by the distributed lasso with the exact wavelet operator, and 0.079 for the denoised estimates produced by the distributed lasso with the approximate wavelet operator with $K = 15$. Note that the approximate solution does not necessarily result in a higher mean square error than the exact solution.

C. Distributed Inverse Filtering

We now consider the situation where node n observes the n^{th} component of $\mathbf{y} = \Psi \mathbf{f} + \boldsymbol{\nu}$, where Ψ is a graph Fourier multiplier operator with multiplier $g_\Psi(\cdot)$, and $\boldsymbol{\nu}$ is uncorrelated Gaussian noise. The task of the network is to recover \mathbf{f} by inverting the effect of the graph multiplier operator Ψ . This is the distributed graph analog to the deblurring problem in imaging, which is discussed in [42, Chapter 7]. As discussed in [42, Chapter 7], trying to recover \mathbf{f} by simply applying the inverse filter in the graph Fourier domain, i.e., setting

$$\begin{aligned} f_*(n) &= \sum_{\ell=0}^{N-1} \left(\frac{1}{g_\Psi(\lambda_\ell)} \right) \hat{y}(\ell) \chi_\ell(n) \\ &= \sum_{\ell=0}^{N-1} \left(\hat{f}(\ell) + \frac{\hat{\nu}(\ell)}{g_\Psi(\lambda_\ell)} \right) \chi_\ell(n) \\ &= f(n) + \sum_{\ell=0}^{N-1} \left(\frac{\hat{\nu}(\ell)}{g_\Psi(\lambda_\ell)} \right) \chi_\ell(n), \end{aligned} \quad (21)$$

does not work well when $g_\Psi(\cdot)$ is zero (or close to zero) for high frequencies, because the summation in (21) blows up, dominating $f(n)$. Therefore, we again use the prior that the signal is smooth with respect to the underlying graph structure, and approximately solve the regularization problem

$$\underset{\mathbf{f}}{\operatorname{argmin}} \frac{\tau}{2} \|\mathbf{y} - \Psi \mathbf{f}\|_2^2 + \mathbf{f}^T \mathcal{L} \mathbf{f} \quad (22)$$

in a distributed manner.

Proposition 6: The solution to (22) is given by $\mathbf{R} \mathbf{y}$, where \mathbf{R} is a graph Fourier multiplier operator with multiplier

$$h(\lambda_\ell) = \frac{\tau g_\Psi(\lambda_\ell)}{\tau g_\Psi^2(\lambda_\ell) + 2\lambda_\ell^r}.$$

The proof of Proposition 6 is included in the Appendix. From Proposition 6, we conclude that one method to perform distributed inverse filtering is to compute $\mathbf{R} \mathbf{y}$, the Chebyshev polynomial approximation to $\mathbf{R} \mathbf{y}$, in a distributed manner according to Algorithm 1.

²The scaling coefficients in the spectral graph wavelet transform are not expected to be sparse.

³The reported errors are averaged over the 441 random graph realizations that were connected.

D. Distributed Semi-Supervised Classification

The goal of *semi-supervised classification* is to learn a mapping from the data points $X = \{x_1, x_2, \dots, x_N\}$ to their corresponding labels $Y = \{y_1, y_2, \dots, y_N\}$. The pairs (x_i, y_i) are independently and identically sampled from a joint distribution $p(x, y)$ over the sample space $\mathcal{X} \times \mathcal{Y}$, where $\mathcal{Y} := \{1, 2, \dots, \kappa\}$ is the space of κ classes. The transductive classification problem is to use the full set of data points $X = \{x_1, x_2, \dots, x_N\}$ and the labels $Y_l = \{y_1, y_2, \dots, y_l\}$ associated with a small portion of the data ($l \ll N$) to predict the labels $Y_u = \{y_{l+1}, y_{l+2}, \dots, y_N\}$ associated with the unlabeled data $X_u = \{x_{l+1}, x_{l+2}, \dots, x_N\}$.

Many semi-supervised learning methods represent the data X by an undirected, weighted graph, and then force the labels to be smooth with respect to the intrinsic structure of this graph. We show how a number of these centralized graph-based semi-supervised classification methods can be distributed using Chebyshev polynomial approximation of generalized graph multiplier operators. Throughout the section, we assume there is one data point at each node in the graph, and the nodes know the weights of the edges connecting them to their neighbors in the graph. For example, each data point could be at a different node in a sensor network, and the weights could be a function of the physical distance between the nodes, as in (1).

1) *Centralized Graph-Based Methods:* For different choices of reproducing kernel Hilbert spaces (RKHS) \mathcal{H} , a number of centralized semi-supervised classification methods estimate the label of the n^{th} data point ($n \in \{l+1, \dots, N\}$) by

$$\arg \max_{j \in \{1, 2, \dots, \kappa\}} F_{nj}^{\text{opt}}, \quad (23)$$

where \mathbf{F}^{opt} is the solution to

$$\mathbf{F}^{\text{opt}} = \arg \min_{\mathbf{F} \in \mathbb{R}^{N \times \kappa}} \sum_{j=1}^{\kappa} \{ \tau \|\mathbf{F}_{:,j} - \mathbf{Y}_{:,j}\|_2^2 + \|\mathbf{F}_{:,j}\|_{\mathcal{H}}^2 \}. \quad (24)$$

In (24), $\mathbf{A}_{:,j}$ denotes the j^{th} column of a matrix \mathbf{A} ; \mathbf{Y} is a $N \times \kappa$ matrix with entries

$$Y_{ij} = \begin{cases} 1, & \text{if } i \in \{1, 2, \dots, l\} \text{ and the label for point } i \text{ is } j, \\ 0, & \text{otherwise} \end{cases}$$

and for some symmetric positive semi-definite matrix $\mathbf{P} \in \mathbb{R}^{N \times N}$,

$$\|\mathbf{f}\|_{\mathcal{H}}^2 = \langle \mathbf{f}, \mathbf{f} \rangle_{\mathcal{H}} := \langle \mathbf{f}, \mathbf{P}\mathbf{f} \rangle = \mathbf{f}^T \mathbf{P}\mathbf{f}. \quad (25)$$

Note that for any symmetric positive semi-definite matrix \mathbf{P} , \mathcal{H} endowed with the inner product defined in (25) is in fact a RKHS on $\mathbf{P}\mathbb{R}^N$, and its kernel is $k(i, j) = (\mathbf{P}^{-1})_{ij}$, where \mathbf{P}^{-1} denotes the pseudoinverse if \mathbf{P} is not invertible [5, Theorem 4].

We now give some examples of graph-based centralized semi-supervised classification methods that fall into this category.

- In Tikhonov regularization, $\mathbf{P} = \mathcal{L}^r$ (e.g., [10])
- Zhou *et al.* [11] take $\mathbf{P} = \mathcal{L}_{\text{norm}}^r$, where $\mathcal{L}_{\text{norm}} := \mathbf{D}^{-\frac{1}{2}} \mathcal{L} \mathbf{D}^{-\frac{1}{2}}$, and also consider a variant with $\mathbf{P} = \mathcal{L} \mathbf{D}^{-1}$

- Smola and Kondor [5] consider a variety of kernel methods, including a diffusion process with $\mathbf{P} = \left[\exp \left(\frac{-\sigma^2}{2\mathcal{L}_{\text{norm}}} \right) \right]^{-1}$, an inverse cosine with $\mathbf{P} = \left[\cos \left(\frac{\pi \mathcal{L}_{\text{norm}}}{4} \right) \right]^{-1}$, and an r -step random walk with $\mathbf{P} = (\sigma \mathbf{I}_N - \mathcal{L}_{\text{norm}})^{-r}$, where $\sigma \geq 2$ and \mathbf{I}_N is the $N \times N$ identity matrix⁴
- Zhu *et al.* [13, Chapter 15] take the kernel approach a step further by solving a convex optimization problem to find a good \mathbf{P}
- Ando and Zhang's K-scaling method [14], [15] takes

$$\mathbf{P} = (\gamma \mathbf{I}_N + \mathbf{D})^{-\frac{1}{2}} (\gamma \mathbf{I}_N + \mathcal{L}) (\gamma \mathbf{I}_N + \mathbf{D})^{-\frac{1}{2}},$$

which reduces to $\mathcal{L}_{\text{norm}}$ when $\gamma = 0$

Before moving on to the distributed semi-supervised classification problem, we note two important properties of the matrices \mathbf{P} used in each of these above methods: 1) They have the same sparsity pattern as \mathcal{L} , and 2) they are easily computable from \mathcal{L} .

2) *Distributing the Centralized Graph-Based Methods:* Now, \mathbf{F}^{opt} in (24) can be equivalently rewritten as the solution to κ separate minimization problems, with

$$\mathbf{F}_{:,j}^{\text{opt}} = \arg \min_{\mathbf{f} \in \mathbb{R}^N} \{ \tau \|\mathbf{f} - \mathbf{Y}_{:,j}\|_2^2 + \mathbf{f}^T \mathbf{P}\mathbf{f} \}. \quad (26)$$

If $\mathbf{P} = \sum_{\ell=0}^{N-1} g(\lambda_{\ell}) \chi_{\ell} \chi_{\ell}^T$ for some $g(\cdot)$, then to solve (26), we can simply compute $\mathbf{R}\mathbf{Y}_{:,j}$, where \mathbf{R} is a graph Fourier multiplier operator with multiplier $\frac{\tau}{\tau + g(\lambda_{\ell})}$. Otherwise, (26) is essentially of the form of (16), with a different choice of multiplier basis. That is, we can write the solution to (26) as $\mathbf{R}\mathbf{Y}_{:,j}$, where \mathbf{R} is a generalized graph multiplier operator of the form (14), with respect to \mathbf{P} . The multiplier is $\frac{\tau}{\tau + \lambda_{\ell}}$.

Therefore, the following is a method to distribute any of the centralized semi-supervised classification methods that can be written as (23) and (24):

- 1) Node n starts with or computes the entries of n^{th} row of \mathbf{P} (which are easily computable from \mathcal{L} in the cases mentioned above)
- 2) Each node n forms the n^{th} row of \mathbf{Y}
- 3) For every $j \in \{1, 2, \dots, \kappa\}$, the nodes compute $\tilde{\mathbf{F}}_{:,j}^{\text{opt}} := \tilde{\mathbf{R}}\mathbf{Y}_{:,j}$ in a distributed manner via Algorithm 1 (with \mathbf{P} replacing \mathcal{L})
- 4) Each node n with an unlabeled data point computes its label estimate according to $\arg \max_{j \in \{1, 2, \dots, \kappa\}} \tilde{F}_{nj}^{\text{opt}}$

By Propositions 2 and 3, as we increase the Chebyshev approximation order K , the classification results of this distributed method converge to results of the corresponding centralized semi-supervised learning method.

VII. COMPARISON WITH OTHER DISTRIBUTED PROCESSING METHODS

In this section, we compare our proposed method with two variations of Jacobi's iterative method. These variations are not distributed computation methods *per se*, but rather centralized computation methods that can be easily parallelized and

⁴All three of these \mathbf{P} matrices can be written as $\mathbf{P} = \sum_{\ell=0}^{N-1} g(\lambda_{\ell}) \chi_{\ell} \chi_{\ell}^T$ for some $g(\cdot)$, where $\{\chi_{\ell}\}_{\ell=0,1,\dots,N-1}$ are the eigenvectors of $\mathcal{L}_{\text{norm}}$.

that we deem most appropriate for the types of applications mentioned above. We have not included the Gauss-Seidel or conjugate gradient methods, for example, because of the extra synchronicity considerations they would require in a distributed implementation.

A. Jacobi's Iterative Method

For $\mathbf{P} = \mathcal{L}_{norm}$, Zhou *et al.* [11] propose to solve (24) through the iteration

$$\mathbf{F}^{(t+1)} = \frac{1}{1+\tau} \left[(\mathbf{I}_N - \mathbf{P}) \mathbf{F}^{(t)} + \tau \mathbf{Y} \right], \quad (27)$$

where $\mathbf{F}^{(0)}$ is arbitrary (they set it to \mathbf{Y}).⁵ The iteration (27) is in fact just a particular instance of Jacobi's iterative method (see, e.g., [43, Chapter 4]) to solve the set of linear equations

$$(\tau \mathbf{I}_N + \mathbf{P}) \mathbf{F}^{opt} = \tau \mathbf{Y}. \quad (28)$$

We can generalize the application of the Jacobi method to solve (26) for the other matrices \mathbf{P} mentioned above, as follows. We write $\mathbf{P} = \mathbf{P}_D - \mathbf{P}_O$, where \mathbf{P}_D is a diagonal matrix with diagonal entries equal to the diagonal entries of \mathbf{P} , and \mathbf{P}_O has zeros on the diagonal, and off-diagonal entries equal to the negative of the off-diagonal entries of \mathbf{P} . Then the Jacobi iterative method to solve (28) is given by

$$\mathbf{F}^{(t+1)} = (\tau \mathbf{I}_N + \mathbf{P}_D)^{-1} \left[\mathbf{P}_O \mathbf{F}^{(t)} + \tau \mathbf{Y} \right]. \quad (29)$$

Note that for $\mathbf{P} = \mathcal{L}_{norm}$, $\mathbf{P}_D = \mathbf{I}_N$ and $\mathbf{P}_O = \mathbf{I}_N - \mathcal{L}_{norm}$, so (29) reduces to (27).

So one alternative distributed semi-supervised classification method is to compute the iterations (29) in a distributed manner, with each node starting with knowledge of its row of \mathbf{P} and \mathbf{Y} . In fact, the communication cost of one iteration of (29) is the same as the communication cost of one iteration of the distributed computation of $\tilde{\mathbf{R}}\mathbf{Y}$ (lines 6 and 7 of Algorithm 1). We present a numerical example comparing the convergence rates of these methods in Section VII-C.

For graph multiplier operators whose multipliers have the property $g(\lambda_\ell) \neq 0$ for all ℓ , we can generalize the Jacobi method as follows. Suppose we wish to compute $\mathbf{R}\mathbf{y}$, where \mathbf{R} is a graph multiplier operator with respect to \mathbf{P} and with multiplier $g(\cdot)$. This is equivalent to solving the linear system of equations

$$\left(\sum_{\ell=0}^{N-1} \frac{1}{g(\lambda_\ell)} \chi_\ell \chi_\ell^* \right) \mathbf{f} = \mathbf{y}.$$

Define the matrix $\mathbf{Q} := \sum_{\ell=0}^{N-1} \frac{1}{g(\lambda_\ell)} \chi_\ell \chi_\ell^*$ and let $\mathbf{Q} = \mathbf{Q}_D - \mathbf{Q}_O$, where \mathbf{Q}_D is the diagonal matrix with diagonal entries equal to those of \mathbf{Q} . Then the Jacobi iteration is

$$\mathbf{x}^{(t+1)} = \mathbf{Q}_D^{-1} \mathbf{Q}_O \mathbf{x}^{(t)} + \mathbf{Q}_D^{-1} \mathbf{y}. \quad (30)$$

However, one immediate drawback of Jacobi's method, as compared with our proposed method, is that it does not always converge. The iterations in (30) converge for any $\mathbf{x}^{(0)}$ if and only if the spectral radius of $\mathbf{Q}_D^{-1} \mathbf{Q}_O$ is less than one [43,

Theorem 4.1]. One sufficient condition for the latter to be true is that \mathbf{Q} is strictly diagonally dominant, as is the case for example when $\mathbf{P} = \mathcal{L}$ and $g(\lambda_\ell) = \frac{\tau}{\tau + \lambda_\ell}$.

B. Jacobi's Iterative Method with Chebyshev Acceleration

When Jacobi's method does converge, we can accelerate (30) using the following algorithm [44, Algorithm 6.7]. Let ρ be an upper bound on the spectral radius of $\mathbf{Q}_D^{-1} \mathbf{Q}_O$, and define $\xi^{(0)} := 1$, $\xi^{(1)} := \rho$, and $\mathbf{x}^{(1)} := \mathbf{Q}_D^{-1} \mathbf{Q}_O \mathbf{x}^{(0)} + \mathbf{Q}_D^{-1} \mathbf{y}$. Then for $t \geq 1$, let

$$\begin{aligned} \xi^{(t+1)} &= \frac{1}{\frac{2}{\rho \xi^{(t)}} - \frac{1}{\xi^{(t-1)}}}, \text{ and} \\ \mathbf{x}^{(t+1)} &= \frac{2\xi^{(t+1)}}{\rho \xi^{(t)}} \mathbf{Q}_D^{-1} \mathbf{Q}_O \mathbf{x}^{(t)} - \frac{\xi^{(t+1)}}{\xi^{(t-1)}} \mathbf{x}^{(t-1)} \\ &\quad + \frac{2\xi^{(t+1)}}{\rho \xi^{(t)}} \mathbf{Q}_D^{-1} \mathbf{y}. \end{aligned} \quad (31)$$

To distribute (31), each node n must first learn Q_{nn} and the n^{th} row of \mathbf{Q}_O . For example, when $\mathbf{P} = \mathcal{L}_{norm}$ and $g(\lambda_\ell) = \frac{\tau}{\tau + \lambda_\ell}$, as in (27), $Q_{nn} = \frac{\tau+1}{\tau}$ for all n , and the n^{th} row of \mathbf{Q}_O is just $-\frac{1}{\tau}$ times the n^{th} row of \mathcal{L}_{norm} , which the nodes can easily compute in a distributed manner. An additional challenge in a distributed setting may be to calculate the bound ρ .

Finally, note that while this method and our method share the same namesake, the use of the Chebyshev polynomials in the two is different. We use Chebyshev polynomials to approximate the multiplier, whereas this method improves the convergence speed of the Jacobi method by using Chebyshev polynomials to choose the weights it uses to form the iterates in (31) as weighted linear combinations of the iterates in (30). See Section 6.5.6 of [44] for more details.

C. Numerical Comparison

An in-depth comparison of the convergence conditions and rates of these methods with our proposed method is beyond the scope of this paper, but we compare their convergence rates here on a few simple numerical examples. We consider the same random graph shown in Figure 2, and we generate a signal \mathbf{f} on the vertices of the graph with the components of \mathbf{f} independently and identically sampled from a uniform distribution on $[-10, 10]$. For different choices of positive semi-definite matrices $\mathbf{P} \in \mathbb{R}^{500 \times 500}$, we define $\mathbf{y} := (\mathbf{I}_{500} + \frac{1}{\tau} \mathbf{P}) \mathbf{f}$, with $\tau = 0.5$. Then, starting with \mathbf{y} , we iteratively compute an approximation to \mathbf{f} in three different distributable ways: 1) $\tilde{\mathbf{R}}\mathbf{y}$, where $\tilde{\mathbf{R}}$ is the Chebyshev approximation to \mathbf{R} , a generalized graph multiplier operator with respect to \mathbf{P} , and with multiplier $\frac{\tau}{\tau + \lambda_\ell}$; 2) with the Jacobi iteration (29); and 3) with the Jacobi iteration with Chebyshev acceleration (31). Since the communication requirements of our method with Chebyshev approximation order K are equal to the communication requirements of K iterations of the latter two methods, we plot the errors $\|\mathbf{f}^{(K)} - \mathbf{f}\|_2$ (where $\mathbf{f}^{(K)}$ corresponds to $\tilde{\mathbf{R}}\mathbf{y}$ with an order K approximation in the first case or the result of the K^{th} iteration in the latter two cases) on the same axes in Figure 6. We repeat the experiment

⁵In [13, Chapter 11], similar iterative label propagation methods from [8] and [12] are also compared with the method of [11].

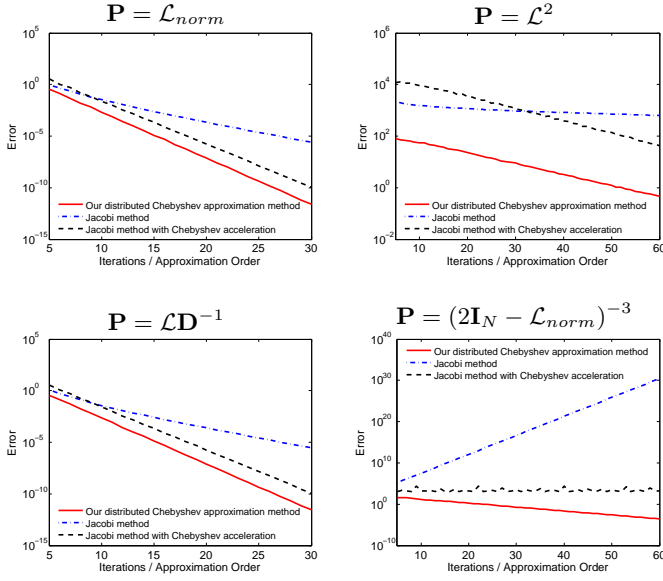


Fig. 6. Three different distributed methods to approximately compute $\mathbf{R}\mathbf{y}$, where \mathbf{R} is a graph multiplier operator with respect to \mathbf{P} for different choices of \mathbf{P} . In all cases, the multiplier is $g(\lambda_\ell) = \frac{\tau}{\tau + \lambda_\ell}$. The error shown is $\|\mathbf{f}^{(K)} - \mathbf{f}\|_2$, where $\mathbf{f}^{(K)}$ is either $\tilde{\mathbf{R}}\mathbf{y}$ with an order K approximation for our distributed Chebyshev approximation method, or the result of the K^{th} iteration for the two Jacobi methods.

with $\mathbf{P} = \mathcal{L}_{\text{norm}}, \mathcal{L}^2, \mathcal{L}\mathbf{D}^{-1}$, and the three-step random walk process $(2\mathbf{I}_{500} - \mathcal{L}_{\text{norm}})^{-3}$, for which the Jacobi method does not converge. In these experiments, not only does our proposed method always converge, but it converges faster than the alternative methods.

VIII. CONCLUDING REMARKS

We presented a novel method to distribute a class of linear operators called unions of graph multiplier operators. The main idea is to approximate the graph multipliers by Chebyshev polynomials, whose recurrence relations make them readily amenable to distributed computation. Key takeaways from the discussion and application examples include:

- A number of distributed signal processing tasks can be represented as distributed applications of unions of graph multiplier operators (and their adjoints) to signals on weighted graphs. Examples include distributed smoothing, denoising, inverse filtering, and semi-supervised learning
- Graph Fourier multiplier operators are the graph analog of filter banks, as they reshape functions' frequencies through multiplication in the Fourier domain
- The amount of communication required to perform the distributed computations only scales with the size of the network through the number of edges of the communication graph, which is usually sparse. Therefore, the method is well suited to large-scale networks
- The approximate graph multiplier operators closely approximate the exact operators in practice, and for graph multiplier operators with smooth multipliers, an upper bound on the spectral norm of the difference of the

approximate and exact operators decreases rapidly as we increase the Chebyshev approximation order

In addition to considering more applications, our ongoing work includes analyzing robustness issues that arise in real networks. For instance, we would like to incorporate quantization and communication noise into the sensor network model, in order to see how these propagate when using the Chebyshev polynomial approximation approach to distributed signal processing tasks. It is also important to analyze the effects of a sensor node dropping out of the network or communicating nodes losing synchronicity to ensure that the proposed method is stable to these disturbances.

IX. APPENDIX

Proof of Proposition 2: For $j = 1, 2, \dots, \eta$ and $n = 1, 2, \dots, N$,

$$\begin{aligned}
 & (\Phi \mathbf{f} - \tilde{\Phi} \mathbf{f})_{(j-1)N+n} \\
 &= \sum_{\ell=0}^{N-1} (g_j(\lambda_\ell) - p_j^K(\lambda_\ell)) \hat{f}(\ell) \chi_\ell(n) \\
 &\leq B(K) \sum_{\ell=0}^{N-1} |\hat{f}(\ell) \chi_\ell(n)| \\
 &\leq B(K) \left(\sum_{\ell=0}^{N-1} |\hat{f}(\ell)|^2 \right)^{\frac{1}{2}} \left(\sum_{\ell=0}^{N-1} |\chi_\ell(n)|^2 \right)^{\frac{1}{2}} \quad (32) \\
 &\leq B(K) \|\mathbf{f}\|_2, \quad (33)
 \end{aligned}$$

where (32) follows from Hölder's inequality, and (33) follows from the Parseval relation (just after equation (49) in [27]) and footnote 3 in [27]. Then

$$\begin{aligned}
 \|\Phi \mathbf{f} - \tilde{\Phi} \mathbf{f}\|_2 &= \sqrt{\sum_{j=1}^{\eta} \sum_{i=1}^N (\Phi \mathbf{f} - \tilde{\Phi} \mathbf{f})_{(j-1)N+n}^2} \\
 &\stackrel{(33)}{\leq} \sqrt{\sum_{j=1}^{\eta} \sum_{i=1}^N [B(K)]^2 \|\mathbf{f}\|_2^2} \\
 &= B(K) \|\mathbf{f}\|_2 \sqrt{\eta N}. \quad (34)
 \end{aligned}$$

Rearranging (34) yields (15). \blacksquare

Proof of Proposition 4: The objective function in (16) is convex in \mathbf{f} . Differentiating it with respect to \mathbf{f} , any solution \mathbf{f}_* to

$$\mathcal{L}^r \mathbf{f}_* + \frac{\tau}{2} (\mathbf{f}_* - \mathbf{y}) = 0 \quad (35)$$

is a solution to (16).⁶ Taking the graph Fourier transform of (35) yields

$$\begin{aligned}
 \widehat{\mathcal{L}^r \mathbf{f}_*}(\ell) + \frac{\tau}{2} (\hat{f}_*(\ell) - \hat{y}(\ell)) &= 0, \quad (36) \\
 \forall \ell \in \{0, 1, \dots, N-1\}.
 \end{aligned}$$

From the real, symmetric nature of \mathcal{L} and the definition of the Laplacian eigenvectors ($\mathcal{L}\chi_\ell = \lambda_\ell \chi_\ell$), we have:

$$\widehat{\mathcal{L}^r \mathbf{f}_*}(\ell) = \chi_\ell^* \mathcal{L}^r \mathbf{f}_* = (\mathcal{L}^r \chi_\ell)^* \mathbf{f}_* = \lambda_\ell^r \chi_\ell^* \mathbf{f}_* = \lambda_\ell^r \hat{f}_*(\ell). \quad (37)$$

⁶In the case $r = 1$, the optimality equation (35) corresponds to the optimality equation in [17, Section III-A] with $p = 2$ in that paper.

Substituting (37) into (36) and rearranging, we have

$$\hat{f}_*(\ell) = \frac{\tau}{\tau + 2\lambda_\ell^r} \hat{y}(\ell), \quad \forall \ell \in \{0, 1, \dots, N-1\}. \quad (38)$$

Finally, taking the inverse graph Fourier transform of (38), we have

$$f_*(n) = \sum_{\ell=0}^{N-1} \hat{f}_*(\ell) \chi_\ell(n) = \sum_{\ell=0}^{N-1} \left[\frac{\tau}{\tau + 2\lambda_\ell^r} \right] \hat{y}(\ell) \chi_\ell(n),$$

$$\forall n \in \{1, 2, \dots, N\}.$$

Proof of Proposition 5: The solutions \mathbf{a}_* to (17) and $\tilde{\mathbf{a}}_*$ to (19) are not unique; however, their images $\Xi^* \mathbf{a}_*$ and $\tilde{\Xi}^* \tilde{\mathbf{a}}_*$ are unique. To see this, for example for $\Xi^* \mathbf{a}_*$, we can write (17) equivalently as

$$\begin{aligned} \underset{\mathbf{a}, \mathbf{b}}{\operatorname{argmin}} \quad & \frac{1}{2} \|\mathbf{y} - \mathbf{b}\|_2^2 + \|\mathbf{a}\|_{1,\mu} \\ \text{s.t.} \quad & \mathbf{b} = \Xi^* \mathbf{a}. \end{aligned}$$

Then by the strict convexity of $\|\cdot\|_2^2$, the convexity of $\|\cdot\|_{1,\mu}$, and Lemma 1 below, $\Xi^* \mathbf{a}_*$ is unique.

Lemma 1: Let $f_1 : \mathbb{R}^n \rightarrow \mathbb{R}$ be strictly convex, $f_2 : \mathbb{R}^m \rightarrow \mathbb{R}$ be convex, and $\mathbf{A} \in \mathbb{R}^{n \times m}$. Then the solution $(\mathbf{x}^*, \mathbf{y}^*)$ to

$$\begin{aligned} \underset{\mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^m}{\operatorname{argmin}} \quad & f_1(\mathbf{x}) + f_2(\mathbf{y}) \\ \text{s.t.} \quad & \mathbf{x} = \mathbf{A}\mathbf{y} \end{aligned} \quad (39)$$

is unique with respect to \mathbf{x}^* (but not necessarily \mathbf{y}^*).

Proof of Lemma 1: Let $(\mathbf{x}_1, \mathbf{y}_1)$ and $(\mathbf{x}_2, \mathbf{y}_2)$ be in the set (39), and assume $\mathbf{x}_1 \neq \mathbf{x}_2$. Then by linearity, $(\mathbf{x}_3, \mathbf{y}_3) := \frac{1}{2}(\mathbf{x}_1, \mathbf{y}_1) + \frac{1}{2}(\mathbf{x}_2, \mathbf{y}_2)$ satisfies $\mathbf{x}_3 = \mathbf{A}\mathbf{y}_3$, and by the strict convexity of $f_1(\cdot)$ and convexity of $f_2(\cdot)$,

$$\begin{aligned} f_1(\mathbf{x}_3) + f_2(\mathbf{y}_3) &< \frac{1}{2}f_1(\mathbf{x}_1) + \frac{1}{2}f_1(\mathbf{x}_2) + \frac{1}{2}f_2(\mathbf{y}_1) + \frac{1}{2}f_2(\mathbf{y}_2) \\ &= \min_{\{\mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^m: \mathbf{x} = \mathbf{A}\mathbf{y}\}} f_1(\mathbf{x}) + f_2(\mathbf{y}), \end{aligned}$$

which is a contradiction. Thus, $\mathbf{x}_1 = \mathbf{x}_2$. ■

It follows from the first-order necessary and sufficient optimality equations of the lasso problem (see, e.g., [39, Proposition 5.3(iv)]) that for all $\mathbf{a} \in \mathbb{R}^{N(J+1)}$, we have

$$\langle \mathbf{y} - \Xi^* \mathbf{a}_*, \Xi^* \mathbf{a} - \Xi^* \mathbf{a}_* \rangle + \|\mathbf{a}_*\|_{1,\mu} \leq \|\mathbf{a}\|_{1,\mu}, \quad (40)$$

and similarly

$$\langle \mathbf{y} - \tilde{\Xi}^* \tilde{\mathbf{a}}_*, \tilde{\Xi}^* \mathbf{a} - \tilde{\Xi}^* \tilde{\mathbf{a}}_* \rangle + \|\tilde{\mathbf{a}}_*\|_{1,\mu} \leq \|\mathbf{a}\|_{1,\mu}. \quad (41)$$

Taking $\mathbf{a} = \tilde{\mathbf{a}}_*$ in (40) and $\mathbf{a} = \mathbf{a}_*$ in (41), summing (40) and (41), and rearranging, we have

$$\begin{aligned} & \langle \mathbf{y} - \Xi^* \mathbf{a}_*, \Xi^* \tilde{\mathbf{a}}_* - \Xi^* \mathbf{a}_* \rangle + \langle \mathbf{y} - \tilde{\Xi}^* \tilde{\mathbf{a}}_*, \tilde{\Xi}^* \mathbf{a}_* - \tilde{\Xi}^* \tilde{\mathbf{a}}_* \rangle \\ &= \|\mathbf{y} - \Xi^* \mathbf{a}_*\|_2^2 + \langle \mathbf{y} - \Xi^* \mathbf{a}_*, \Xi^* \tilde{\mathbf{a}}_* - \mathbf{y} \rangle \\ &+ \|\mathbf{y} - \tilde{\Xi}^* \tilde{\mathbf{a}}_*\|_2^2 + \langle \mathbf{y} - \tilde{\Xi}^* \tilde{\mathbf{a}}_*, \tilde{\Xi}^* \mathbf{a}_* - \mathbf{y} \rangle \leq 0. \end{aligned} \quad (42)$$

Then

$$\begin{aligned} & \|\tilde{\Xi}^* \tilde{\mathbf{a}}_* - \Xi^* \mathbf{a}_*\|_2^2 \\ &= \|\mathbf{y} - \Xi^* \mathbf{a}_*\|_2^2 + \|\mathbf{y} - \tilde{\Xi}^* \tilde{\mathbf{a}}_*\|_2^2 - 2\langle \mathbf{y} - \Xi^* \mathbf{a}_*, \mathbf{y} - \tilde{\Xi}^* \tilde{\mathbf{a}}_* \rangle \\ &\stackrel{(42)}{\leq} \langle \mathbf{y} - \Xi^* \mathbf{a}_*, (\tilde{\Xi}^* - \Xi^*) \tilde{\mathbf{a}}_* \rangle + \langle \mathbf{y} - \tilde{\Xi}^* \tilde{\mathbf{a}}_*, (\Xi^* - \tilde{\Xi}^*) \mathbf{a}_* \rangle \\ &\leq \|\mathbf{y} - \Xi^* \mathbf{a}_*\|_2 \|\tilde{\Xi}^* - \Xi^*\|_2 \|\tilde{\mathbf{a}}_*\|_2 \\ &\quad + \|\mathbf{y} - \tilde{\Xi}^* \tilde{\mathbf{a}}_*\|_2 \|\Xi^* - \tilde{\Xi}^*\|_2 \|\mathbf{a}_*\|_2 \\ &\leq \|\mathbf{y}\|_2 \|\tilde{\Xi} - \Xi\|_2 (\|\tilde{\mathbf{a}}_*\|_2 + \|\mathbf{a}_*\|_2), \end{aligned} \quad (43)$$

where (43) follows from the Cauchy-Schwarz inequality, and (44) follows from the facts that $\|A^*\|_2 = \|A\|_2$ [35, p. 309], and $\|\mathbf{y} - \Xi^* \mathbf{a}_*\|_2 \leq \|\mathbf{y}\|_2$ and $\|\mathbf{y} - \tilde{\Xi}^* \tilde{\mathbf{a}}_*\|_2 \leq \|\mathbf{y}\|_2$ by the optimality of \mathbf{a}_* and $\tilde{\mathbf{a}}_*$, and the feasibility of $\mathbf{a} = \mathbf{0}$. Finally, by the uniqueness of $\Xi^* \mathbf{a}_*$, $\|\mathbf{a}_*\|_{1,\mu}$ is the same for all solutions \mathbf{a}_* , and

$$\begin{aligned} & \left\{ \min_i \mu_i \right\} \|\mathbf{a}_*\|_2 \\ &\leq \|\mathbf{a}_*\|_{1,\mu} \leq \frac{1}{2} \|\mathbf{y} - \Xi^* \mathbf{a}_*\|_2^2 + \|\mathbf{a}_*\|_{1,\mu} \leq \frac{1}{2} \|\mathbf{y}\|_2^2, \end{aligned} \quad (45)$$

where the last inequality again follows from feasibility of $\mathbf{a} = \mathbf{0}$. The bound in (45) also holds for $\{\min_i \mu_i\} \|\tilde{\mathbf{a}}_*\|_2$, and substituting these into (44) yields the desired result. ■

Proof of Proposition 6: As in Proposition 4, the objective function in (22) is convex in \mathbf{f} . Differentiating it with respect to \mathbf{f} , any solution \mathbf{f}_* to

$$\mathcal{L}^r \mathbf{f}_* + \frac{\tau}{2} \Psi^* (\Psi \mathbf{f}_* - \mathbf{y}) = 0 \quad (46)$$

is a solution to (22). Note that

$$\widehat{\Psi^* \Psi \mathbf{f}_*}(\ell) = g_\Psi^2(\lambda_\ell) \hat{f}_*(\ell), \quad (47)$$

and

$$\widehat{\Psi^* \mathbf{y}}(\ell) = g_\Psi(\lambda_\ell) \hat{y}(\ell). \quad (48)$$

Therefore, taking the graph Fourier transform of (46) and substituting in (37), (47), and (48) yields

$$\hat{f}_*(\ell) = \frac{\tau g_\Psi(\lambda_\ell)}{\tau g_\Psi^2(\lambda_\ell) + 2\lambda_\ell^r} \hat{y}(\ell), \quad \forall \ell \in \{0, 1, \dots, N-1\}.$$

REFERENCES

- [1] M. Rabbat and R. Nowak, "Distributed optimization in sensor networks," in *Proc. Int. Symp. Inf. Process. Sensor Netw.*, Berkeley, CA, Apr. 2004, pp. 20–27.
- [2] J. B. Predd, S. R. Kulkarni, and H. V. Poor, "Distributed learning in wireless sensor networks," *IEEE Signal Process. Mag.*, vol. 23, pp. 56–69, Jul. 2006.
- [3] R. Olfati-Saber, J. Fax, and R. Murray, "Consensus and cooperation in networked multi-agent systems," *Proc. IEEE*, vol. 95, no. 1, pp. 215–233, Jan. 2007.
- [4] A. G. Dimakis, S. Kar, J. M. F. Moura, M. G. Rabbat, and A. Scaglione, "Gossip algorithms for distributed signal processing," *Proc. IEEE*, vol. 98, no. 11, pp. 1847–1864, Nov. 2010.
- [5] A. J. Smola and R. Kondor, "Kernels and regularization on graphs," in *Proc. Ann. Conf. Comp. Learn. Theory*, ser. Lect. Notes Comp. Sci., B. Schölkopf and M. Warmuth, Eds. Springer, 2003, pp. 144–158.
- [6] D. Zhou and B. Schölkopf, "A regularization framework for learning from graph data," in *Proc. ICML Workshop Stat. Relat. Learn. and Its Connections to Other Fields*, Jul. 2004, pp. 132–137.

- [7] —, “Regularization on discrete spaces,” in *Pattern Recogn.*, ser. Lect. Notes Comp. Sci., W. G. Kropatsch, R. Sablatnig, and A. Hanbury, Eds. Springer, 2005, vol. 3663, pp. 361–368.
- [8] X. Zhu and Z. Ghahramani, “Learning from labeled and unlabeled data with label propagation,” Carnegie Mellon University, Technical Report CMU-CALD-02-107, 2002.
- [9] —, “Semi-supervised learning using Gaussian fields and harmonic functions,” in *Proc. Int. Conf. Mach. Learn.*, Washington, D.C., Aug. 2003, pp. 912–919.
- [10] M. Belkin, I. Matveeva, and P. Niyogi, “Regularization and semi-supervised learning on large graphs,” in *Learn. Theory*, ser. Lect. Notes Comp. Sci. Springer-Verlag, 2004, pp. 624–638.
- [11] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf, “Learning with local and global consistency,” in *Adv. Neural Inf. Process. Syst.*, S. Thrun, L. Saul, and B. Schölkopf, Eds., vol. 16. MIT Press, 2004, pp. 321–328.
- [12] O. Delalleau, Y. Bengio, and N. Le Roux, “Efficient non-parametric function induction in semi-supervised learning,” in *Proc. Int. Wkshp. on Artif. Intell. Stat.*, Barbados, Jan. 2005, pp. 96–103.
- [13] O. Chapelle, B. Schölkopf, and A. Zien, Eds., *Semi-Supervised Learning*. MIT Press, 2006.
- [14] R. K. Ando and T. Zhang, “Learning on graph with Laplacian regularization,” in *Adv. Neural Inf. Process. Syst.*, B. Schölkopf, J. Platt, and T. Hofmann, Eds., vol. 19. MIT Press, 2007, pp. 25–32.
- [15] R. Johnson and T. Zhang, “On the effectiveness of Laplacian normalization for graph semi-supervised learning,” *J. Mach. Learn. Res.*, vol. 8, pp. 1489–1517, 2007.
- [16] S. Bogleux, A. Elmoataz, and M. Melkemi, “Discrete regularization on weighted graphs for image and mesh filtering,” in *Scale Space Var. Methods Comp. Vision*, ser. Lect. Notes Comp. Sci., F. Sgallari, A. Murli, and N. Paragios, Eds. Springer, 2007, vol. 4485, pp. 128–139.
- [17] A. Elmoataz, O. Lezoray, and S. Bogleux, “Nonlocal discrete regularization on weighted graphs: a framework for image and manifold processing,” *IEEE Trans. Image Process.*, vol. 17, pp. 1047–1060, Jul. 2008.
- [18] F. Zhang and E. R. Hancock, “Graph spectral image smoothing using the heat kernel,” *Pattern Recogn.*, vol. 41, pp. 3328–3342, Nov. 2008.
- [19] G. Peyré, S. Bogleux, and L. Cohen, “Non-local regularization of inverse problems,” in *Proc. ECCV’08*, ser. Lect. Notes Comp. Sci., D. A. Forsyth, P. H. S. Torr, and A. Zisserman, Eds. Springer, 2008, pp. 57–68.
- [20] L. Rosasco, E. De Vito, and A. Verri, “Spectral methods for regularization in learning theory,” DISI, Università degli Studi di Genova, Italy, Technical Report DISI-TR-05-18, 2005.
- [21] R. Wagner, V. Delouille, and R. Baraniuk, “Distributed wavelet denoising for sensor networks,” in *Proc. IEEE Int. Conf. Dec. and Contr.*, San Diego, CA, Dec. 2006, pp. 373–379.
- [22] S. Barbarossa, G. Scutari, and T. Battisti, “Distributed signal subspace projection algorithms with maximum convergence rate for sensor networks with topological constraints,” in *Proc. IEEE Int. Conf. Acc., Speech, and Signal Process.*, Taipei, Apr. 2009, pp. 2893–2896.
- [23] C. Guestrin, P. Bodik, R. Thibaux, M. Paskin, and S. Madden, “Distributed regression: an efficient framework for modeling sensor network data,” in *Proc. Int. Symp. Inf. Process. Sensor Netw.*, Berkeley, CA, Apr. 2004, pp. 1–10.
- [24] J. B. Predd, S. R. Kulkarni, and H. V. Poor, “A collaborative training algorithm for distributed learning,” *IEEE Trans. Inf. Theory*, vol. 55, no. 4, pp. 1856–1871, Apr. 2009.
- [25] J. A. Bazerque, G. Mateos, and G. B. Giannakis, “Distributed lasso for in-network linear regression,” in *Proc. IEEE Int. Conf. Acc., Speech, and Signal Process.*, Dallas, TX, Mar. 2010, pp. 2978–2981.
- [26] G. Mateos, J.-A. Bazerque, and G. B. Giannakis, “Distributed sparse linear regression,” *IEEE Trans. Signal Process.*, vol. 58, no. 10, pp. 5262–5276, Oct. 2010.
- [27] D. K. Hammond, P. Vandergheynst, and R. Gribonval, “Wavelets on graphs via spectral graph theory,” *Appl. Comput. Harmon. Anal.*, vol. 30, no. 2, pp. 129–150, Mar. 2011.
- [28] D. I. Shuman, P. Vandergheynst, and P. Frossard, “Chebyshev polynomial approximation for distributed signal processing,” in *Proc. Int. Conf. Distr. Comput. in Sensor Syst.*, Barcelona, Spain, June 2011.
- [29] F. K. Chung, *Spectral Graph Theory*. Vol. 92 of the CBMS Regional Conference Series in Mathematics, AMS Bookstore, 1997.
- [30] T. Birykoğlu, J. Leydold, and P. F. Stadler, *Laplacian Eigenvectors of Graphs*. Lecture Notes in Mathematics, vol. 1915, Springer, 2007.
- [31] J. C. Mason and D. C. Handscomb, *Chebyshev Polynomials*. Chapman and Hall, 2003.
- [32] G. M. Phillips, *Interpolation and Approximation by Polynomials*. CMS Books in Mathematics, Springer-Verlag, 2003.
- [33] T. J. Rivlin, *Chebyshev Polynomials*. Wiley-Interscience, 1990.
- [34] W. N. Anderson and T. D. Morley, “Eigenvalues of the Laplacian of a graph,” *Linear Multilinear Algebra*, vol. 18, no. 2, pp. 141–145, 1985.
- [35] R. A. Horn and C. R. Johnson, *Matrix Analysis*. Cambridge University Press, 1990.
- [36] R. Tibshirani, “Regression shrinkage and selection via the Lasso,” *J. Royal. Statist. Soc. B*, vol. 58, no. 1, pp. 267–288, 1996.
- [37] S. Chen, D. Donoho, and M. Saunders, “Atomic decomposition by basis pursuit,” *SIAM J. Sci. Comp.*, vol. 20, no. 1, pp. 33–61, Aug. 1998.
- [38] I. Daubechies, M. Defrise, and C. De Mol, “An iterative thresholding algorithm for linear inverse problems with a sparsity constraint,” *Commun. Pure Appl. Math.*, vol. 57, no. 11, pp. 1413–1457, Nov. 2004.
- [39] P. L. Combettes and V. R. Wajs, “Signal recovery by proximal forward-backward splitting,” *Multiscale Model. Sim.*, vol. 4, no. 4, pp. 1168–1200, Nov. 2005.
- [40] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, 1989.
- [41] D. K. Hammond, “Spectral graph wavelets toolbox.” [Online]. Available: <http://wiki.epfl.ch/sgwt>
- [42] G. Peyré, *Advanced Signal, Image and Surface Processing*, 2010, <http://www.ceremade.dauphine.fr/~peyre/numerical-tour/book/>.
- [43] Y. Saad, *Iterative Methods for Sparse Linear Systems*. PWS Publishing Company, 1996.
- [44] J. W. Demmel, *Applied Numerical Linear Algebra*. SIAM, 1997.